

A COMPUTER CODE FOR SOLVING MEDIUM SIZED
NON-LINEAR PROGRAMMING PROBLEMS BY THE
METHOD OF FEASIBLE DIRECTIONS

James Douglas Harrison

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93940

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

A COMPUTER CODE FOR SOLVING MEDIUM SIZED
NON-LINEAR PROGRAMMING PROBLEMS BY THE
METHOD OF FEASIBLE DIRECTIONS

by

James Douglas Harrison

September 1974

Thesis Advisor:

G. T. Howard

Approved for public release; distribution unlimited.

T163068

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Computer Code for Solving Medium Sized Non-Linear Programming Problems by the Method of Feasible Directions		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; September 1974
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) James Douglas Harrison		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE September 1974
		13. NUMBER OF PAGES 70
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Non-Linear Programming Feasible Directions Computer-Code Maximization Constrained Optimization		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A computer code, FEASBL, is developed to maximize a non-linear objective function over a convex feasible region. The feasible region is defined by a set of non-linear and linear constraints on the variables. FEASBL can solve problems involving up to fifty variables with a feasible region formed by up to fifty non-linear		

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

Block #20 continued

constraints, and fifty linear constraints. FEASBL uses a feasible direction method as its solution algorithm.

A Computer Code for Solving Medium Sized
Non-Linear Programming Problems by the
Method of Feasible Directions

by

James Douglas Harrison
Lieutenant Commander, United States Navy
B.Chem., University of Minnesota, 1963

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1974

ABSTRACT

A computer code, FEASBL, is developed to maximize a non-linear objective function over a convex feasible region. The feasible region is defined by a set of non-linear and linear constraints on the variables. FEASBL can solve problems involving up to fifty variables with a feasible region formed by up to fifty non-linear constraints, and fifty linear constraints. FEASBL uses a feasible direction method as its solution algorithm.

TABLE OF CONTENTS

I.	INTRODUCTION.....	7
II.	MATHEMATICAL BACKGROUND.....	10
	A. DEFINITIONS.....	10
	B. THE METHOD.....	11
	1. The Direction Finding Problem.....	11
	2. The Modified Direction Finding Problem...	13
	3. The Step Length Problem.....	14
	4. Stopping Conditions.....	15
	C. THE INITIAL FEASIBLE POINT.....	15
III.	THE COMPUTER CODE FEASBL.....	18
IV.	USING FEASBL.....	25
	A. INPUT.....	23
	1. The FUNCTION Routines.....	23
	2. Data Input.....	26
	3. Data Limits.....	27
	4. The Program CHECK.....	28
	5. System Cards.....	28
	a. FEASBL with Source Decks.....	29
	b. FEASBL with Object Decks.....	30
	c. CHECK with Source Decks.....	31
	d. CHECK with Object Decks.....	32
	B. SAMPLE PROBLEM.....	32
V.	EVALUATION.....	38
	COMPUTER PROGRAM FEASBL.....	40

COMPUTER PROGRAM CHECK..... 64

LIST OF REFERENCES..... 69

INITIAL DISTRIBUTION LIST..... 70

I. INTRODUCTION

This thesis presents a computer code for solving medium sized non-linear programming problems, and describes its use at the Naval Postgraduate School computer center.

The general non-linear programming problem is the optimization of an objective function over a constrained feasible region. By the technique of separable programming, many non-linear problems can be converted to a linear problem and solved using established linear programming methods. Many problems, however, cannot be solved by this method. Several techniques have been developed for solving non-linear problems. Most methods involve following the gradient of the objective function to generate a sequence of points that converges to the optimal point. The method used in this code is Zoutendijk's method of feasible directions [1]. The method of feasible directions can be considered a large-step gradient method as opposed to the small-step gradient and search methods. Feasible direction methods can be characterized by:

1. Starting at an initial feasible point.
2. Establishing a feasible direction that will improve the objective function.
3. Proceeding to a new feasible point that has a better value of the objective function than the previous point.
4. Converging to the optimal point, or determining that the problem is unbounded.

Many procedures can be considered feasible directions methods, for example, the simplex method for solving linear programming problems.

The method used in this code requires that the user know the objective function, constraint functions, all their first partial derivatives, and an initial feasible point. If the user does not know an initial feasible point, this code can be used to solve a preliminary programming problem which will provide a feasible point. This problem will be described later. This code will solve maximization problems of the form:

$$\begin{aligned} \text{Max:} \quad & z = F(\underline{x}) \\ \text{Subj to:} \quad & G_i(\underline{x}) \leq b_i, \quad i \in I_C \\ & a_i^T \underline{x} \leq b_i, \quad i \in I_L \\ & d_j \leq x_j \leq c_j, \quad j \in J. \end{aligned}$$

The functions $F(\underline{x})$, and $G_i(\underline{x})$ are non-linear functions which are supplied by the user as Fortran FUNCTION routines. The sets I_C and I_L are the sets of subscripts for the non-linear, and linear constraints respectively. If the objective function is concave, and the non-linear constraints are convex functions, then the problem will be a convex programming problem, and the method will converge to the global maximum. If the problem is not convex, then the method may converge to a local maximum. The code is designed to accommodate functions with up to fifty variables. The constraint set

can include up to fifty non-linear constraints and fifty linear constraints. A complete description of how to use this code is provided in Section IV.

II. MATHEMATICAL BACKGROUND

The theoretical foundation for the method of feasible directions was developed by Zoutendijk in 1960 [1]. A brief discussion of the mathematical background as it applies to the method used in this computer code is presented.

A. DEFINITIONS

A region R is a convex region if for \underline{x}_1 and \underline{x}_2 , two points in R , then the point $\underline{x}_3 = a\underline{x}_1 + (1 - a)\underline{x}_2$ is in R for all a such that $0 \leq a \leq 1$.

Let R be the feasible region, and if \underline{x} is a member of R then a direction \underline{s} is a feasible direction if and only if there exists a $\lambda' > 0$ such that $\underline{x} + \lambda \underline{s}$ is in R for all $\lambda \in (0, \lambda')$.

A usable feasible direction is a feasible direction \underline{s} such that $\nabla_{\underline{x}} F(\underline{x})^T \underline{s} > 0$. In other words, a usable feasible direction at \underline{x} is a direction in which the value of the objective function increases and along which it is possible to move while remaining in R .

Let $F(\underline{x})$ be a concave function, and let the feasible region R be convex. Then if there exists no usable feasible direction at \underline{x} in R , then \underline{x} must be a maximum of $F(\underline{x})$ over the region R . To show this assume that \underline{x} is not a maximum and there is no usable feasible direction at \underline{x} . There exists a point \underline{y} in R such that $F(\underline{y}) > F(\underline{x})$ and since R is convex all the points along the line between \underline{y} and \underline{x} must be in R .

Further, since $F(\underline{x})$ is concave, $\nabla_{\underline{x}} F(\underline{x})^T (\underline{y} - \underline{x}) > 0$, thus the direction $\underline{y} - \underline{x}$ is a usable feasible direction. This contradiction implies that \underline{x} must be a maximum of $F(\underline{x})$.

B. THE METHOD

The method of feasible directions begins at an initial feasible point \underline{x}^0 , determines the "best" usable feasible direction, moves along that direction to the point \underline{x}^k that maximizes $F(\underline{x})$ along the direction without leaving the feasible region, repeats this procedure until a point \underline{x}^* is found which has no usable direction, or determines that the problem is unbounded. The method can thus be viewed as a four step procedure.

1. Determine a usable feasible direction \underline{s}^k at the point \underline{x}^k .
2. If no usable feasible direction exists stop, \underline{x}^k is the maximum of $F(\underline{x})$ over R .
3. Determine a step length λ_k such that $\underline{x}^{k+1} = \underline{x}^k + \lambda_k \underline{s}^k$ is in R and \underline{x}^{k+1} maximizes $F(\underline{x})$ along \underline{s}^k .
4. If λ_k goes to infinity, then the problem is unbounded, otherwise go back to step 1 and repeat.

The direction finding problem will be discussed first, and then the step length problem.

1. The Direction Finding Problem

The direction finding problem is to find the usable feasible direction that will result in the greatest apparent increase in the objective function. This can be formulated as the following linear programming problem.

Max: σ

Subj to: $\sigma - \nabla_x F(\underline{x})^T \underline{s} \leq 0$

$\sigma + \nabla_x G_i(\underline{x})^T \underline{s} \leq 0, i \in I_c(\underline{x})$

$\underline{a}_i^T \underline{s} \leq 0, i \in I_1(\underline{x})$

$s_j \geq 0, j \in J^-(\underline{x})$

$s_j \leq 0, j \in J^+(\underline{x})$

$|s_j| \leq 1, j \in J(\underline{x}),$

where $I_c(\underline{x})$ and $I_1(\underline{x})$ are the sets of binding non-linear constraints and binding linear constraints, and $J^-(\underline{x})$ and $J^+(\underline{x})$ are the sets of j such that x_j is at its lower and upper bound respectively.

If there exists at least one point that is strictly interior to R with respect to the non-linear constraints and if the optimal value of σ is 0 in the direction finding problem, then there is no usable feasible direction and we are at the optimum point for the non-linear problem. The last constraint in the direction finding problem is a normalization constraint. In order to insure that the procedure will converge the components of \underline{s}^k must be bounded, and several normalizations have been studied by Zoutendijk. These normalizations are:

N1: $\underline{s}^T \underline{s} \leq 1$

N2: $|s_j| \leq 1$

$$N3: \quad s_j \leq 1 \quad \text{if } \nabla_x F(\underline{x})_j > 0$$

$$s_j \geq -1 \quad \text{if } \nabla_x F(\underline{x})_j < 0$$

$$N4: \quad \sigma \leq 1$$

$$N5: \quad \nabla_x G_i(\underline{x})^T \underline{s} + \sigma \leq b_i - G_i(\underline{x}); \quad i \in I_c(\underline{x})$$

$$\underline{a}_i^T \underline{s} \leq b_i - \underline{a}_i^T \underline{x}; \quad i \in I_L(\underline{x})$$

$$d_j - x_j \leq s_j \leq c_j - x_j.$$

The normalization N1 will generally lead to fewer steps than N2, N3, or N4, but the amount of computation for each step is considerably greater. Also the simple constraints $s_j \leq 0$, and $s_j \geq 0$ increases the size of the direction finding problem if N1 is used, but not for N2, N3, and N4. As one goes from N2, to N4 the number of steps needed increases, but the amount of computation for each step decreases. N5 leads to a larger problem, and also does not converge as fast as the other normalizations. Normalization N2 was chosen for this code as the best compromise between the number of steps and the amount of work for each step.

2. The Modified Direction Finding Problem

To insure that the procedure actually converges to the maximum, some precautions must be taken to prevent the phenomenon of "jamming" in which the procedure in effect gets "stuck in a corner" and never finds the real maximum. This is accomplished by considering nearby constraints in addition to the binding constraints when solving the direction finding problem.

Define the sets:

$$I_C(\underline{x}, \epsilon) = \{i \in I_C : b_i - \epsilon \leq G_i(\underline{x}) \leq b_i\}$$

$$I_L(\underline{x}, \epsilon) = \{i \in I_L : b_i - \epsilon \leq \underline{a}_i^T \underline{x} \leq b_i\}$$

$$J^-(\underline{x}, \epsilon) = \{j \in J : d_j \leq x_j \leq d_j + \epsilon\}$$

$$J^+(\underline{x}, \epsilon) = \{j \in J : c_j - \epsilon \leq x_j \leq c_j\}.$$

The direction finding problem is then modified to include the constraints in the sets $I_C(\underline{x}, \epsilon)$, $I_L(\underline{x}, \epsilon)$, $J^-(\underline{x}, \epsilon)$, and $J^+(\underline{x}, \epsilon)$. Therefore the direction finding problem becomes:

$$\text{Max: } \sigma$$

$$\text{Subj to: } \sigma - \nabla_x F(\underline{x})^T \underline{s} \leq 0$$

$$+ \nabla_x G_i(\underline{x})^T \leq 0, \quad i \in I_C(\underline{x}, \epsilon)$$

$$\underline{a}_i^T \underline{s} \leq 0, \quad i \in I_L(\underline{x}, \epsilon)$$

$$s_j \geq 0, \quad j \in J^-(\underline{x}, \epsilon)$$

$$s_j \leq 0, \quad j \in J^+(\underline{x}, \epsilon)$$

$$|s_j| \leq 1, \quad j \in J(\underline{x}).$$

Then if the solution to the direction finding problem is such that $\sigma < \epsilon$, the problem is solved again with $\epsilon/2$. Thus we stay away from the "corners" until we get near the optimal point.

3. The Step Length Problem

Once the direction \underline{s}^k has been found, the problem is to determine how far to move in that direction. If λ_F is defined to be the largest step length λ such that $\underline{x}^k + \lambda \underline{s}^k$

is in R , then the step length will be λ_k such that $F(\underline{x}^k + \lambda_k \underline{s}^k)$ is the maximum of $F(\underline{x}^k + \lambda \underline{s}^k) : \lambda \leq \lambda_F$. If $\lambda_k = \infty$, then the problem is unbounded, and we can conclude that the maximum of $F(\underline{x})$ is infinite. The maximum feasible step λ_F is found by a simple search along the direction \underline{s}^k from the point \underline{x}^k . Then the optimal step length λ_k is determined by finding the λ such that $\nabla_{\underline{x}} F(\underline{x}^k + \lambda \underline{s}^k) \underline{s}^k = 0$. Since $F(\underline{x})$ is concave, if $\nabla_{\underline{x}} F(\underline{x}^k + \lambda_F \underline{s}^k)^T \underline{s}^k > 0$, then λ_F is used for the step length, otherwise λ_k can be found by *regula falsi* [2].

4. Stopping Conditions

The procedure will terminate whenever the solution to the direction finding problem yields $\sigma = 0$, indicating that there are no usable feasible directions. If the solution to the step length problem is $\lambda_k = \infty$, then the problem is unbounded and the procedure is terminated indicating an unbounded solution. In addition, if the step length or the percent improvement in the objective function becomes smaller than user provided limits, the problem is terminated, and the current solution is specified.

C. THE INITIAL FEASIBLE POINT

If an initial feasible point is not known, pick an \underline{x}' such that $d_j \leq x'_j \leq c_j$, and solve the problem:

$$\begin{aligned} \text{Max:} & \quad -z_1 \\ \text{Subj to:} & \quad \underline{a}_i^T \underline{x} - p_i z_1 \leq b_i ; i \in I_L \\ & \quad d_j \leq x_j \leq c_j \end{aligned}$$

$$z_1 \geq 0$$

where

$$p_i = \begin{cases} 0 & ; \text{ if } \underline{a}_i^T \underline{x}' \leq b_i \\ \underline{a}_i^T \underline{x}' - b_i & ; \text{ if } \underline{a}_i^T \underline{x}' > b_i. \end{cases}$$

An initial feasible point for this linear programming problem is \underline{x}' , $z_1 = 1$. If the feasible region is non-empty, then the optimal solution of this problem will be $z_1 = 0$, $\underline{x} = \underline{x}''$. Now \underline{x}'' will be feasible with respect to the linear constraints, if \underline{x}'' is also feasible with respect to the non-linear constraints, then use \underline{x}'' for the initial feasible point. However if \underline{x}'' is not feasible with respect to the non-linear constraints, solve the problem:

$$\text{Max:} \quad -z_2$$

$$\text{Subj to:} \quad G_i(\underline{x}) - p_i z_2 \leq b_i \quad ; i \in I_C$$

$$\underline{a}_i^T \underline{x} \leq b_i \quad ; i \in I_L$$

$$d_j \leq x_j \leq c_j$$

$$z_2 \geq 0$$

where:

$$p_i = \left\{ \begin{array}{ll} 0 & ; \text{ if } G_i(\underline{x}'') \leq b_i \\ G_i(\underline{x}'') - b_i & ; \text{ if } G_i(\underline{x}'') > b_i. \end{array} \right\}$$

This non-linear programming problem can be solved by the method of feasible directions using \underline{x}'' , $z_2 = 1$ as the initial feasible point. If the feasible region of the original problem is non-empty, then the optimal solution to this problem will be $z_2 = 0$, and $\underline{x} = \underline{x}^0$. Now the original problem can be solved using \underline{x}^0 as the initial feasible point.

III. THE COMPUTER PROGRAM FEASBL

This section will describe the computer programming aspects of the code, a complete set of user instructions is provided in section IV. Several programming languages were considered for the program including IBM 360/FORTRAN IV, SIMSCRIPT II.5, and PL/1. FORTRAN IV was chosen because its virtually universal availability and widespread familiarity will allow easy installation at other computer centers, and facilitate user modifications in the future.

FEASBL is structured as a straight line program, consisting of several sections; input, direction finding problem building and solving, step length determination, and output. In deciding the size of the problem that the program would be able to handle, the size of the direction finding linear programming problem was the controlling factor. To insure that round-off errors would not degrade the solution, the inverse of the basis matrix is maintained in double precision. FEASBL is capable of handling fifty variables, with fifty non-linear and fifty linear constraints thus requiring a basis matrix that is 152 by 152, which requires over 184K of core. Desiring to keep the total size below 300K if possible, substantial effort was made to minimize the storage requirements throughout the program both by not storing any value that could be computed when needed, and using the same storage space for multiple purposes. The code now requires

about 290K of core space in addition to that required for user provided functions.

The information required by the code includes the problem description, the initial starting point, and some program control variables. The problem description is provided by specifying the objective function, non-linear and linear constraints, upper and lower bounds on the variables, and the requirements vectors. In addition, the first partial derivatives of the objective function and the non-linear constraint functions are required. A prototype code was developed which did not require derivatives, but which severely limited the form of the functions that could be used. It was felt, however, that the greatly increased power and flexibility gained by not restricting the form of the functions more than offset the requirement to provide the derivatives. These functions are provided in the form of Fortran FUNCTION routines, thus FEASBL can handle any function the user can express as a Fortran routine. The remaining data is input in the Fortran NAMELIST format. This method was chosen because the unstructured format minimizes keypunch errors.

Each iteration begins by building the direction finding problem. The sets $I_C(\underline{x}, \epsilon)$, $I_L(\underline{x}, \epsilon)$, $J^-(\underline{x}, \epsilon)$ and $J^+(\underline{x}, \epsilon)$ are determined, and then the direction finding linear programming problem is built. Rather than store the entire A matrix, only those elements which depend on the gradients of the objective function and non-linear constraints in $I_C(\underline{x}, \epsilon)$ referred to as the core are stored. The linear programming problem is

solved using the revised simplex algorithm [3], and the basis inverse is maintained through the product form. At each step, every non-basic variable is priced, and that variable with the most negative $z_j - c_j$ is selected for entry. In the event of ties the left-most variable is picked. The column vector $\underline{a}_{j,k}$ for the entering variable is then generated, and used to compute $\underline{y}_k = B^{-1} \cdot \underline{a}_k$. The departing variable is j such that $x_{b_j}/y_{j,k}$ is minimum for all $y_{j,k} > 0$. If there are no negative $z_j - c_j$'s, then the optimal solution has been found for the direction finding problem. After an optimal solution has been found, the basis matrix is created and inverted directly using the IBM subroutine DMINV, and the solution is checked for optimality. If the solution is not optimal, then the linear program is continued until an optimal solution is reached. When the optimal solution is achieved, the direction is available, and the program proceeds to find the best step length. This is accomplished by first finding λ_F , the largest λ such that $\underline{x}^k + \lambda \underline{s}^k$ is feasible, and then determining $\lambda_k \leq \lambda_F$ that maximizes $F(\underline{x}^k + \lambda_k \underline{s}^k)$. The quantity λ_F is found by starting with $\lambda = 1$ and increasing λ by a factor of 2 until a λ_{NF} is found such that $\underline{x}^k + \lambda_{NF} \underline{s}^k$ is not feasible. The interval between λ_{NF} and the largest known feasible step λ_F^1 is reduced by checking $\lambda = \frac{1}{2}(\lambda_{NF} + \lambda_F^1)$ for feasibility until the interval is less than the minimum step length (STPMIN). However, if a feasible step length λ is found such that $F(\underline{x}^k + \lambda \underline{s}^k) < F(\underline{x}^k)$, then it is assumed that the maximum point along the direction

lies between these two points. Once λ_F has been determined λ_k is computed by the method of *regula falsi*. Since $F(\underline{x})$ is concave and is increasing at \underline{x}^k along the direction \underline{s}^k the directional derivative $(\nabla_{\underline{x}} F(\underline{x})^T \underline{s})$ is positive at \underline{x}^k and will be zero at the maximum along \underline{s}^k . If the directional derivative is negative at $\underline{x}^k + \lambda_F \underline{s}^k$, then the maximum lies between the two points. A new λ is then selected such that $\lambda = (b\lambda_1 - a\lambda_2) / (b - a)$ where: $a = \nabla_{\underline{x}} F(\underline{x}^k + \lambda_1 \underline{s}^k)$; $b = \nabla_{\underline{x}} F(\underline{x}^k + \lambda_2 \underline{s}^k)$; and λ_1 and λ_2 bracket the optimal stop length. If the directional derivative is zero at the new point the maximum has been found, otherwise we have a new interval with positive and negative directional derivatives at the end-points, and another λ is determined. This continues until either the maximum is found, or the interval becomes less than the minimum specified. If the directional derivative is positive at $\underline{x}^k + \lambda_F \underline{s}^k$ and $F(\underline{x}^k + \lambda_F \underline{s}^k) > F(\underline{x}^k)$, then the maximum along \underline{s}^k is outside the feasible region, and λ_k is set equal to λ_F . However, if $F(\underline{x}^k + \lambda_F \underline{s}^k) \leq F(\underline{x}^k)$, then the function is not concave, so a step λ_U between 0 and λ_F is found such that the directional derivative is negative, and it is assumed that $F(\underline{x})$ is concave between \underline{x}^k and $\underline{x}^k + \lambda_U \underline{s}^k$, and the maximum is sought as before.

Once a usable feasible direction, and step length have been determined set $\underline{x}^{k+1} = \underline{x}^k + \lambda_k \underline{s}^k$ and repeat the process starting at \underline{x}^{k+1} . This process is continued until either no usable feasible direction can be found or the problem becomes unbounded. If the step length or the percent

improvement in the objective function become less than user supplied limits the program will terminate. The program will also stop after a user specified number of iterations.

IV. USING FEASBL

This section will describe how to use FEASBL, including required input, possible errors, and the normal output. An example is given illustrating different features of the code.

A. INPUT

The objective function, non-linear constraint functions, and their first partial derivatives are input as Fortran FUNCTION routines. The linear constraint matrix, requirements vectors, upper and lower bounds on x_j , the initial feasible point, and any other input is read in as data at execution.

1. The FUNCTION Routines

Four Fortran FUNCTION routines are required, these routines provide the objective function, its first partial derivatives, the non-linear constraints, and their first partial derivatives. Even if the problem has no non-linear constraints, all four FUNCTION routines must be provided.

The four routines are:

<u>FUNCTION</u>	<u>Definition</u>
FX(X)	The objective function.
DELFX(NX, X)	The partial derivatives of the objective function.
GIX(NGI, X)	The non-linear constraint functions.
DELGIX(NX,NGI,X)	The partial derivatives of the non-linear constraints.

The arguments of the FUNCTIONS statements are:

<u>Argument</u>	<u>Definition</u>
X	The vector \underline{x} .
NX	Subscript of the partial derivative desired, i.e. $\partial F(\underline{x})/\partial x_{NX}$.
NGI	The non-linear constraint number.

Each routine must include a statement dimensioning X(e.g.; REAL*4 X(8), or DIMENSION X(15)), at least one assignment statement with the name of the FUNCTION on the left of the equal sign (e.g.; FX = X(1), or DELGIX = 4.0 * X(2)**3), and at least one RETURN statement. No variation in the names of the FUNCTIONS, nor in the order of the arguments is allowed. The following example illustrates the use of the FUNCTION routines.

$$\begin{array}{ll}
 \text{Max:} & F(\underline{x}) = 6x_1 + 8x_2 - x_1^2 - x_2^2 \\
 \text{Subj to:} & \left. \begin{array}{l} 4x_1^2 + x_2^2 \leq 16 \\ (4 - x_1)^2 + x_2^2 \leq 16 \end{array} \right\} \text{non-linear constraints} \\
 & \left. \begin{array}{l} 3x_1 + 5x_2 \leq 15 \\ 2x_1 + 12x_2 \leq 30 \end{array} \right\} \text{linear constraints} \\
 & 0 \leq x_1 \leq 1.85 \\
 & 2.0 \leq x_2.
 \end{array}$$

For this problem the following FUNCTION routines are required:

- a. The objective function

```
FUNCTION FX( X )  
REAL*4 X(2)  
FX = 6.0 * X(1) + 8.0 * X(2) - X(1)**2 - X(2)**2  
RETURN  
END
```

- b. The partial derivatives of the objective function

```
FUNCTION DELFX( NX, X )  
REAL*4 X(2)  
GO TO ( 101, 102 ), NX  
101 DELFX = 6.0 - 2.0 * X(1)  
RETURN  
102 DELFX = 8.0 - 2.0 * X(2)  
RETURN  
END
```

- c. The non-linear constraint functions

```
FUNCTION GIX( NGI, X )  
REAL*4 X(2)  
GO TO ( 100, 200 ), NGI  
100 GIX = 4.0 * X(1)**2 + X(2)**2  
RETURN  
200 GIX = ( 4.0 - X(1) )**2 + X(2)**2  
RETURN  
END
```

- d. The partial derivatives of the non-linear constraints

```
FUNCTION DELGIX( NX, NGI, X )  
REAL*4 X(2)  
GO TO ( 100, 200 ), NGI  
100 GO TO ( 101, 102 ), NX  
101 DELGIX = 8.0 * X(1)  
RETURN  
102 DELGIX = 2.0 * X(2)
```



```

      RETURN
200 GO TO ( 201, 202 ), NX
201 DELGIX = 2.0 * X(1) - 8.0
      RETURN
202 DELGIX = 2.0 * X(2)
      RETURN
      END

```

The significant feature to note is that when a FUNCTION is called by FEASBL specifying a point X, and when necessary the partial derivative desired NX, and the constraint number NGI, the appropriate value is returned.

2. Data Input

The remainder of the data needed is read in from data cards at execution in NAMELIST format. This format was chosen to minimize keypunch errors, and provide flexibility for the user. The following variables are required:

<u>Name</u>	<u>Description</u>
IXMAX	Number of variables.
IGCNT	Number of non-linear constraints.
ILCNT	Number of linear constraints.

The following variables are optional, and if not provided by the user will take on the default values listed.

<u>Name</u>	<u>Default</u>	<u>Description</u>
GL(I,J)	0.0	Linear constraint matrix element
XMIN(I)	0.0	Lower bound on x_j
XMAX(I)	$\approx 10^{75}$	Upper bound on x_i
BC(I)	0.0	The b_i for the non-linear constraints.

<u>Name</u>	<u>Default</u>	<u>Description</u>
BL(I)	0.0	The b_i for the linear constraints.
X(I)	0.0	The initial feasible point.
EPSLON	0.1	The initial anti-jamming constant.
STPMIN	10^{-3}	The minimum step length.
PERCNT	10^{-2}	The minimum objective function improvement.
ITMAX	25	The maximum number of iterations.

The first data card begins with an ampersand (&) in column two followed by the word INPUT with no space (&INPUT). The values are then listed as "variable name = FORTRAN constant" i.e. IXMAX = 2, GL(1,1) = 3.0, the last data item is followed by an ampersand and the word "END" (&END). If more than one card is required, each card must begin in column 2 or more, and the &END will appear on the last card only. For the above example the data cards would look like:

```
&INPUT IXMAX = 2, IGCNT=2, ILCNT=2, GL(1,1)=3.0, GL(1,2)=5.0,
GL(2,1)=2.0, GL(2,2)=12.0, XMAX(1)=1.85, XMIN(2)=2.0,
BC(1)=16.0, BC(2)=16.0, BL(1)=15.0, BL(2)=30.0, ITMAX=10,
EPSLON=0.01, STPMIN=1.0E-04, X(1)=1.0, X(2)=2.0, &END
```

Here the order is unimportant, and all the variables need not be listed. In this example the default values for XMIN(1), XMAX(2), and PERCNT will be used since they are not listed on the data cards.

3. Data Limits

Input errors will result if the variable inputs are outside the following limits:

$$1 \leq \text{IXMAX} \leq 50$$

$$0 \leq \text{IGCNT} \leq 50$$

$$0 \leq \text{ILCNT} \leq 50$$

$$\text{XMIN(I)} < \text{XMAX(I)}$$

To prevent stopping prematurely, EPSLON should be significantly larger than STPMIN.

After all the data is input, the initial point is checked for feasibility. If the point is not feasible the program is terminated.

4. The Program CHECK

FEASBL requires 290K of core space in addition to that required for the FUNCTION routines. Most problems will require less than 320K and less than 4 minutes to run. As an aid for checking the input data and FUNCTION routines, the program CHECK can be used. CHECK requires exactly the same input as FEASBL, and performs the same input error checks. In addition CHECK will determine the feasibility of the initial point and print out the value of the objective function, all the constraints, and the value of all the partial derivatives of the objective function and non-linear constraints. By using CHECK the user can quickly verify the input, since CHECK can be run normally as a class B or C job, before using FEASBL, which is a class J job.

5. System Cards

The JCL cards required to use FEASBL or CHECK depend on the way that the FUNCTION routines are input. The routines

can be input as either Fortran source decks and compiled as part of the job, or the routines can be precompiled and input as object decks. An example of each method follows.

a. FEASBL with Source Decks

```
// standard OS JOB card           ,TIME=(03,59)
// EXEC FORTCLG,REGION.GO=300K
//FORT.SYSIN DD *
        (FUNCTION source decks)
/*
//LINK.NLPLIB DD DSNAME=F0600.NLPLIB,UNIT=2321,
// VOL=SER=CEL001,DISP=SHR,LABEL=(, , ,IN)
//LINK.SYSIN DD *
        INCLUDE NLPLIB(FEASBL,DARRAY)
/*
//GO.SYSIN DD *
        &INPUT IXMAX=2, IGCNT=2, ... , &END
/*
```


b. FEASBL with Object Decks

```
//      standard OS JOB card           ,TIME=(03,59)
// EXEC LGO,REGION.GO=300K
//LINK.NLPLIB DD DSNAME=F0600.NLPLIB,UNIT=2321,
//      VOL=SER=CEL001,DISP-SHR,LABEL=(,,,IN)
//LINK.SYSIN DD *
        INCLUDE NLPLIB(FEASBL,DARRAY)
        (FUNCTION object decks)
/*
//GO.SYSIN DD *
        &INPUT IXMAX=2, IGCNT=2, ... , X(2)=2.0, &END
/*
```


c. CHECK with Source Decks

```
//      standard OS JOB card
// EXEC FORTCLG,REGION.GO=100K
//FORT.SYSIN DD *
        (FUNCTION source decks)
/*
//LINK.NLPLIB DD DSNAME=F0600.NLPLIB,UNIT=2321,
//  VOL=SER=CEL001,DISP=SHR,LABEL=(,,IN)
//LINK.SYSIN DD *
        INCLUDE NLPLIB(CHECK)
/*
//GO.SYSIN DD *
        &INPUT IXMAX=2, IGCNT=2, ... ,X(2)=2.0, &END
/*
```


d. CHECK with Object Decks

```
//      standard OS JOB card
// EXEC LGO,REGION.GO=100K
//LINK.NLPLIB DD DSN=NAME=F0600.NLPLIB,UNIT=2321,
// VOL=SER=CEL001,DISP=SHR,LABEL=(, , ,IN)
//LINK.SYSIN DD *
        INCLUDE NLPLIB(CHECK)
        (FUNCTION objective decks)
/*
//GO.SYSIN DD *
        &INPUT IXMAX=2, IGCNT=2, ... , X(2)=2.0, &END
/*
```

A set of object decks can be produced either after checking the FUNCTION routines by executing a FORTCD job, or in conjunction with the program CHECK by replacing // EXEC FORTCLG,REGION.GO=100K with // EXEC FORTCLGP,PARM.FORT='DECK',REGION.GO=100K.

B. SAMPLE PROGRAM

The FUNCTION routines, input cards, and system cards required to use FEASBL with source decks are shown for the problem:

$$\begin{array}{ll} \text{Max:} & F(\underline{x}) = 6x_1 + 8x_2 - x_1^2 - x_2^2 \\ \text{Subj to:} & 4x_1^2 + x_2^2 \leq 16 \\ & (4 - x_1)^2 + x_2^2 \leq 16 \end{array}$$

$$3x_1 + 5x_2 \leq 15$$

$$2x_1 + 12x_2 \leq 30$$

$$0 \leq x_1 \leq 1.85$$

$$2.0 \leq x_2 \leq \infty$$

A sample of the output of both CHECK and FEASBL for this problem is provided. The next two pages are the cards required for executing FEASBL with the sample problem. This is followed by the output of CHECK and then FEASBL for the sample problem.


```
//SAMPLE01 JOB (1115,0601,ROY2),'SAMPLE JOB',TIME=(03,59)
// EXEC FORTCLG,REGION.GO=300K
//FORT.SYSIN DD *
FUNCTION FX( X )
```

```

C      THIS FUNCTION PROVIDES THE OBJECTIVE FUNCTION.
C
C      REAL*4 X(2)
```

```

C      THIS IS THE OBJECTIVE FUNCTION
C
C      FX = 6.0 * X(1) + 8.0 * X(2) - X(1)**2 - X(2)**2
C      RETURN
C      END
```

```
*****
```

```
FUNCTION DELFX( NX, X )
```

```
THIS FUNCTION PROVIDES THE FIRST PARTIAL DERIVATIVES
OF THE OBJECTIVE FUNCTION.
```

```
REAL*4 X(2)
GO TO ( 101, 102 ), NX
```

```
THIS IS THE PARTIAL WITH RESPECT TO X(1).
```

```
101 DELFX = 6.0 - 2.0 * X(1)
RETURN
```

```
THIS IS THE PARTIAL WITH RESPECT TO X(2)
```

```
102 DELFX = 8.0 - 2.0 * X(2)
RETURN
END
```

```
*****
```

```
FUNCTION GIX( NGI, X )
```

```
THIS FUNCTION PROVIDES THE NON-LINEAR CONSTRAINTS.
```

```
REAL*4 X(2)
```

```
GO TO THE PROPER CONSTRAINT.
```

```
GO TO ( 100, 200 ), NGI
```

```
THIS IS NON-LINEAR CONSTRAINT NUMBER 1.
```

```
100 GIX = 4.0 * X(1)**2 + X(2)**2
RETURN
```

```
THIS IS NON-LINEAR CONSTRAINT NUMBER 2.
```

```
200 GIX = ( 4.0 - X(1) )**2 + X(2)**2
RETURN
END
```

```
*****
```

```
FUNCTION DELGIX( NX, NGI, X )
```

```
THIS FUNCTION PROVIDES THE FIRST PARTIAL DERIVATIVES
OF THE NON-LINEAR CONSTRAINT FUNCTIONS.
```



```

C      REAL*4 X(2)
C      GO TO THE CONSTRAINT SPECIFIED.
C      GO TO ( 100, 200 ),NGI
C      THIS IS CONSTRAINT NUMBER ONE, WHICH PARTIAL IS WANTED
C 100 GO TO ( 101, 102 ), NX
C      THIS IS THE PARTIAL WITH RESPECT TO X(1).
C 101 DELGIX = 8.0 * X(1)
C      RETURN
C      THIS IS THE PARTIAL WITH RESPECT TO X(2)
C 102 DELGIX = 2.0 * X(2)
C      RETURN
C      THIS IS CONSTRAINT NUMBER TWO, WHICH PARTIAL IS WANTED
C 200 GO TO ( 201, 202 ), NX
C      THIS IS THE PARTIAL WITH RESPECT TO X(1)
C 201 DELGIX = 2.0 * X(1) - 8.0
C      RETURN
C      THIS IS THE PARTIAL WITH RESPECT TO X(2)
C 202 DELGIX = 2.0 * X(2)
C      RETURN
C      END

```

```

//LINK.NLPLIB DD DSN=F0600.NLPLIB,UNIT=2321,
//      VOL=SER=CELO01,DISP=SHR,LABEL=(,,IN)
//LINK.SYSIN DD *
//      INCLUDE NLPLIB(FEASBL,DARRAY)
//GO.SYSIN DD *
&INPUT IXMAX=2, IGCNT=2, ILCNT=2, GL(1,1)=3.0, GL(1,2)=5.0,
GL(2,1)=2.0, GL(2,2)=12.0, XMAX(1)=1.85, XMIN(2)=2.0,
BC(1)=16.0, BC(2)=16.0, BL(1)=15.0, BL(2)=30.0, ITMAX=10,
EPSLGN=0.01, STPMIN=1.0E-04, X(1)=1.0, X(2)=2.0, &END

```


SAMPLE OUTPUT FROM CHECK

NON-LINEAR PROGRAMMING PROBLEM

INITIAL SET UP:

NUMBER OF VARIABLES: 2
 NUMBER OF NON-LINEAR CONSTRAINTS: 2
 NUMBER OF LINEAR CONSTRAINTS: 2
 THE LINEAR CONSTRAINT MATRIX IS:
 (1, 1); 3.00000 (1, 2); 5.00000
 (2, 1); 2.00000 (2, 2); 12.00000

THE BOUNOS ON THE VARIABLES ARE:

0.0 < X(1) < 1.85000 ; 2.00000 < X(2) < 0.723701E 70

NON-LINEAR CONSTRAINTS REQUIREMENTS VECTOR, BC(I):

(1); 16.0000 (2); 16.0000 (

LINEAR CONSTRAINTS REQUIREMENTS VECTOR, BL(I):

(1); 15.0000 (2); 30.0000 (

THE INITIAL FEASIBLE STARTING POINT IS:

X(1); 1.00000 ,X(2); 2.00000 ,X(

ANTI JAMMING CONSTANT: 0.100000E-01

MINIMUM STEP LENGTH: 0.100000E-03

MINIMUM IMPROVEMENT: 0.100000E-01

MAXIMUM # ITERATIONS: 10

INPUT COMPLETE. INITIAL VALUE OF OBJECTIVE FUNCTION: 1.700000E 01

VALUE OF NON-LINEAR CONSTRAINT NUMBER, 1, IS: 8.0000000E 00

VALUE OF NON-LINEAR CONSTRAINT NUMBER, 2, IS: 1.3000000E 01

THE VALUE OF LINEAR CCNSTRAINT NUMBER 1, IS: 1.3000000E 01

THE VALUE OF LINEAR CCNSTRAINT NUMBER 2, IS: 2.6000000E 01

THE BOUNOS, AND ALL THE NON-LINEAR AND LINEAR CONSTRAINTS HAVE BEEN CHECKED.

NEXT THE VALUES OF THE PARTIAL OERIVATIVES OF THE OBJECTIVE FUNCTION AND THE CONSTRAINTS ARE LISTED.

THE VALUE OF OELFX(1, X) IS: 4.000000E 00

THE VALUE OF OELFX(2, X) IS: 4.000000E 00

THE VALUE OF OELGIX(1, 1, X) IS: 8.000000E 00

THE VALUE OF OELGIX(2, 1, X) IS: 4.000000E 00

THE VALUE OF OELGIX(1, 2, X) IS: -6.000000E 00

THE VALUE OF OELGIX(2, 2, X) IS: 4.000000E 00

SAMPLE OUTPUT FROM FEASBL

NON-LINEAR PROGRAMMING PROBLEM

INITIAL SET UP:

NUMBER OF VARIABLES: 2
 NUMBER OF NON-LINEAR CONSTRAINTS: 2
 NUMBER OF LINEAR CONSTRAINTS: 2
 THE LINEAR CCNSTRAINT MATRIX IS:
 (1, 1); 3.00000 (1, 2); 5.00000
 (2, 1); 2.00000 (2, 2); 12.0000

THE BCUNOS ON THE VARIABLES ARE:

0.0 < X(1) < 1.85000 ; 2.00000 < X(2) < 0.723701E 76
 NON-LINEAR CONSTRAINTS REQUIREMENTS VECTOR, BC(I):
 (1); 16.0000 (2); 16.0000 (1);
 LINEAR CONSTRAINTS REQUIREMENTS VECTOR, BL(I):
 (1); 15.0000 (2); 30.0000 (1);

THE INITIAL FEASIBLE STARTING POINT IS:

X(1); 1.00000 ,X(2); 2.00000 ,X(1);
 ANTI JAMMING CONSTANT: 0.100000E-01
 MINIMUM STEP LENGTH: 0.100000E-03
 MINIMUM IMPROVEMENT: 0.100000E-01
 MAXIMUM # ITERATIONS: 10

INPUT COMPLETE. INITIAL VALUE OF OBJECTIVE FUNCTION: 1.700000E 01

PROBLEM SOLUTION

ITERATION NUMBER: 1
 F(X) = 1.867500E 01; IMPROVEMENT THIS ITERATION: 11.0294 PERCENT.
 THE POINT X() IS:

X(1), 1.250000 ; X(2), 2.250000 ; X(1)

ITERATION NUMBER: 2
 F(X) = 1.922217E 01; IMPROVEMENT THIS ITERATION: 1.8393 PERCENT.
 THE POINT X() IS:

X(1), 1.666625 ; X(2), 2.000024 ; X(1)

OPTIMAL SOLUTION:

THE OPTIMAL VALUE OF THE OBJECTIVE FUNCTION IS: 1.922217E 01

THE POINT X() IS:

X(1), 1.666625 ; X(2), 2.000024 ; X(1)

V. EVALUATION

To validate the code, and evaluate its performance, several non-linear problems were solved with FEASBL. These problems were selected to represent the range of problems FEASBL was designed to solve. The problems included unconstrained optimization of a concave objective function, constrained optimization of concave objective functions some with the unconstrained maxima interior and some exterior to the feasible regions. An unbounded problem was also solved.

In each case the optimum solution was obtained in less than five seconds of cpu time. These problems were all small in order to facilitate manual verification of the solutions. Larger problems will take correspondingly more time, but the code appears to be quite efficient. In most problems the optimal solution was obtained after two iterations. In one case, the solution took seven iterations.

It was found that the value of STPMIN is critical to the performance of FEASBL. If STPMIN is set too small, when the interval between λ_F and λ_{NF} is calculated, the round-off error may exceed the value of STPMIN. This results in the code looping indefinitely trying to reduce the interval beyond machine accuracy leading to a program completion due to excess time. If this should occur the problem should be restarted at the last point with a larger value for STPMIN. With step lengths on the order of 1.0 the interval cannot be

determined to an accuracy greater than 10^{-6} . With larger step lengths this value will increase. An initial value of 10^{-2} or 10^{-3} is recommended for STPMIN. This value can be reduced in subsequent runs of FEASBL if a more precise solution is desired.

The initial value of EPSLON is not as critical as STPMIN. EPSLON should be greater than STPMIN but generally not greater than 10^{-1} . Experience with FEASBL indicates that values on the order of 10^{-1} or 10^{-2} perform well. If a large value is used, on the order of 1.0 or greater, and an optimal point is found, this should be verified using a smaller value for EPSLON, as the code may stop early with large values of EPSLON.

Performance was unaffected by variations in PERCNT. Large values may cause the program to terminate sooner. Values less than 10^{-6} are equivalent to zero because of inherent machine accuracy.

An attempt was made to solve a problem in which all the constraints were equality constraints, thus the feasible region had no strictly interior points. The program would terminate immediately without any significant movement. This phenomenon was anticipated since the underlying theory assumes that the feasible region contains at least one strictly interior point to the set of non-linear constraints [1].

INT00490	INT00500	INT00510	INT00520	INT00530	INT00540	INT00550	INT00560	INT00570	INT00580	INT00590	INT00600	INT00610	INT00620	INT00630	INT00640	INT00650	INT00660	INT00670	INT00680	INT00690	INT00700	INT00710	INT00720	INT00730	INT00740	INT00750	INT00760	INT00770	INT00780	INT00790	INT00800	INT00810	INT00820	INT00830	INT00840	INT00850	INT00860	INT00870	INT00880	INT00890	INT00900	INT00910	INT00920	INT00930	INT00940	INT00950	INT00960	INT00970	INT00980	INT00990
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------

GL(I,J)	LINEAR CONSTRAINT MATRIX ELEMENT.
XXMIN(I)	LOWER BOUND ON X(I).
XMAX(I)	UPPER BOUND ON X(I).
BC(I)	NON-LINEAR CONSTRAINT REQUIREMENT VECTOR ELEMENT.
BL(I)	LINEAR CONSTRAINT REQUIREMENTS VECTOR ELEMENT.
X(I)	THE INITIAL FEASIBLE POINT.
EPSLON	THE INITIAL ANTI-JAMMING CONSTANT.
STPMIN	THE MINIMUM STEP LENGTH.
PERCNT	THE MINIMUM OBJECTIVE FUNCTION IMPROVEMENT.
ITMAX	MAXIMUM NUMBER OF ITERATIONS.

FOR A COMPLETE DESCRIPTION SEE:

```

**      IMPLICIT REAL*4 (A-H,C-Z), INTEGER*2 (I-L), INTEGER*4 (M-N),
**

```

```

LOGICAL *1 ($)
REAL*8 B
DIMENSION ION(100), DSUM, DZJ, DBI, DEL
DIMENSION ION(100), X(152), BC(50), BL(50), PHI(152)
DIMENSION ION(100), XSAVE(50), XTWO(50), S(50), YJD(152), ZJCJ(253)
DIMENSION ION(100), IXIN(50), NWORK(152), IMAK(50)
DIMENSION ION(100), IAN(50), NCRKL(152), IALIM(50)
DIMENSION ION(100), XACRES(100), XMAX(50), XMIN(50)
DIMENSION ION(100), INVRSC(152), AJ(152), X(152), XMAX(50), XMIN(50)
DIMENSION ION(100), XACRES(100), XMAX(50), XMIN(50)

```



```

DIMENSION $BASIC(253), $YJPOS(253)
DIMENSION GL(50,50)
EQUIVALENCE (B(1), BC(1)), (B(51), BL(1))
EQUIVALENCE (YJ(1), PHI(1)), (XB(1), XDIR)
EQUIVALENCE (YJ(1), NWORKE(1)), (AJ(1), NWORKE(1))
EQUIVALENCE (YJ(1), XSAVE(1)), (YJ(51), XONE(1))
EQUIVALENCE (YJ(147), XLAMUP), (YJ(148), XLAMNF)
EQUIVALENCE (YJ(149), XLAMF), (YJ(150), XLAMDA)
EQUIVALENCE (YJ(151), XLINC), (AJ(1), XTWO(1))
EQUIVALENCE (GL(1,1), GLCNST(1,1))
EQUIVALENCE (GL(1,1), ILCNT, GL, XMIN, XMAX,
NAMELIST / INPUT, IXMAX, IGCNT, ILCNT, ITMAX,
1 2C, BL, X, EPSLON, STPMIN, PERCENT, ITMAX

      SET THE INITIAL VALUES

DATA BINVRS / 23104 * 0.000 /
DATA ZJCJ / 253*0.0EO /, 3 / 100*0.0EO /, YJ / 152*0.0EO /
DATA XMIN / 50*0.0EO /, XMAX / 50*Z7FFFFF /, ITER / 1 /
DATA EPSLON / 1.0E-01 /, STPMIN / 1.0E-03 /, PERCENT / 1.0E-02 /
DATA CMAX / Z7FFFFF /, NREAD / 5 /, NWRITE / 6 /
DATA ZCJST / -1.0E-05 /
DATA GLCNST / 2500*0.0EO /, ITMAX / 25 /
DATA X / 50*0.0EO /

INPUT SECTION:
READ IN FOLLOWING VALUES: IXMAX, IGCNT, ILCNT, GLCNST(I,J),
XMIN(I), XMAX(I), BC(I), BL(I), X(I), EPSLON, STPMIN, PERCENT
PRINT OUT ALL INPUT VALUES AND INITIAL VALUE OF OBJECT2VE
FUNCTION.

READ(NREAD, INPUT)
WRITE(NWRITE, 901) IXMAX, IGCNT, ILCNT
FORMAT(1, 50X, 'NON-LINEAR PROGRAMMING PROBLEM', //, 0, 'INITIAL SE
IT UP:', 0, 5X, 'NUMBER OF VARIABLES:', 13X, 15, /, 0, 5X, 'NUMBER OF
2NON-LINEAR CONSTRAINTS:', 15, /, 0, 5X, 'NUMBER OF LINEAR CONSTRAINTS
3:', 4X, 15)

      VERIFY THAT THE PARAMETERS ARE WITHIN LIMITS

IF ( IXMAX .LE. 50 .AND. IGCNT .LE. 50 .AND. ILCNT .LE. 50 ) GO TO
1 1000
WRITE(NWRITE, 902)
902 FORMAT(0 * * * * *
1D * * * * *)
STOP
1000 IF( ILCNT .EQ. 0 ) GO TO 1005
WRITE(NWRITE, 904)
904 FORMAT(0, 5X, 'THE LINEAR CONSTRAINT MATRIX IS:')

```

CC C

CCCCCCCC

CC C


```

DC 1004 I=1, ILCNT
DO 1003 N=1, I XMAX
IF( MOD(N,3) .EQ. 1 ) WRITE(NWRITE,1902) I, N, GL(I,N)
IF( MOD(N,3) .EQ. 2 ) WRITE(NWRITE,1903) I, N, GL(I,N)
IF( MOD(N,3) .EQ. 0 ) WRITE(NWRITE,1904) I, N, GL(I,N)
1902 FORMAT(0,05X,((,I2,),(,I2,),(,G15.6)
1903 FORMAT(+,05X,((,I2,),(,I2,),(,G15.6)
1904 FORMAT(+,03X,((,I2,),(,I2,),(,G15.6)
1003 CONTINUE
WRITE(NWRITE,1901)
FORTE
CONTINUE
1901 CONTINUE
1004 FORMAT(0,05X, THE BOUNDS ON THE VARIABLES ARE:',//)
1909 WRITE(NWRITE,1909)
1005 DO 1018 N=1, I XMAX
IF( MOD(N,2) .EQ. 1 ) WRITE(NWRITE,1912) XMIN(N), N, XMAX(N)
1912 IF( MOD(N,2) .EQ. 1 ) WRITE(NWRITE,1913) XMIN(N), N, XMAX(N)
1913 IF( MOD(N,2) .EQ. 0 ) WRITE(NWRITE,1913) XMIN(N), N, XMAX(N)
1018 FORMAT(+,05X,G15.6, ' < X(',I2,') < ',G15.6)
CONTINUE
C
C
C
CHECK THE UPPER AND LOWER BOUNDS FOR OBVIOUS
ERRORS. STOP IF ANY EXIST.
DO 1019 I=1, I XMAX
IF( XMIN(I) .LT. XMAX(I) ) GO TO 1019
1919 IF( XMIN(I) .LT. XMAX(I) ) GO TO 1019
FORMAT(0 * * * ERROR IN BOUNDS, PROGRAM TERMINATED. * * * *)
STOP
1019 CONTINUE
IF( IGCNT .EQ. 0 ) GO TO 1030
1920 IF( IGCNT .EQ. 0 ) GO TO 1030
WRITE(NWRITE,1921) (I,BC(I), I=1, IGCNT)
1921 WRITE(NWRITE,1921) (I,BC(I), I=1, IGCNT)
1921 FORMAT(0,05X, NON-LINEAR CONSTRAINTS REQUIREMENTS VECTOR, BC(I):',
1921 1,/, (0,05X,4((,I2,),(,I2,),(,G20.6)/))
1030 IF( ILCNT .EQ. 0 ) GO TO 1035
WRITE(NWRITE,1930) (I, BL(I), I=1, ILCNT)
1930 IF( ILCNT .EQ. 0 ) GO TO 1035
1930 FORMAT(0,05X, LINEAR CONSTRAINTS REQUIREMENTS VECTOR, BL(I):',/,
1930 1(0,05X,4((,I2,),(,I2,),(,G20.6)/))
1035 WRITE(NWRITE,1935) (I, X(I), I=1, I XMAX)
1935 WRITE(NWRITE,1935) (I, X(I), I=1, I XMAX)
1935 FORMAT(0,05X, THE INITIAL FEASIBLE STARTING POINT IS:',/,
1935 1(0,05X,4((,I2,),(,I2,),(,G20.6)/))
1937 WRITE(NWRITE,1937) EPSILON, PERCENT, ITMAX
1937 FORMAT(0,05X, ANTI JAMMING CONSTANT: ',G20.6,/, '0',5X, 'MINIMUM
1937 1STEP LENGTH: ',G20.6,/, '0',5X, 'MINIMUM IMPROVEMENT: ',G20.6,
1937 2(0,05X, MAXIMUM # ITERATIONS: ',G20.6)
FOFX = FX(X)
1938 WRITE(NWRITE,1938) FOFX
1938 WRITE(NWRITE,1938) FOFX
1938 FORMAT(0 INPUT COMPLETE. INITIAL VALUE OF OBJECTIVE FUNCTION:',
1938 11PE15.6,/, '1',58X, 'PROBLEM SOLUTION')

```



```

C      X(I) IS WITHIN EPSLON OF LOWER BOUND
C      KOUNT1 = KOUNT1 + 1
C      IMIN(KOUNT1) = I
C      GO TO 2010
C      X(I) IS INTERIOR
C      KOUNT3 = KOUNT3 + 1
C      IXIN(KOUNT3) = I
C      2004
C      2010 CONTINUE
C      STORE NUMBER OF BINDING AND NONBINDING BOUNDS
C      JLO = KOUNT1
C      JHI = KOUNT2
C      JIN = KOUNT3
C      ARE THERE ANY NON-LINEAR CONSTRAINTS
C      IF( IGCNT .LE. 0 ) GO TO 2030
C      RESET THE COUNTER, THEN CHECK THE N.L. CONSTRAINTS
C      KOUNT1 = 0
C      DO 2020 N=1,IGCNT
C      XX = GIX(N,X)
C      XHI = BC(N)
C      XLO = XHI - EPSLON
C      IS X INTERIOR TO CONSTRAINT N
C      IF( XX .LT. XLO ) GO TO 2020
C      IS X INFEASIBLE WITH RESPECT TO CONSTRAINT N
C      IF( XX .GT. XHI ) GO TO 2092
C      X IS WITHIN EPSLON OF CONSTRAINT N
C      KOUNT1 = KOUNT1 + 1
C      IGBIND(KOUNT1) = N
C      GO TO 2020
C      2092
C      WRITE(NWRITE,2992) N, XX
C      2992
C      FORMAT('O * * * * X IS NOT FEASIBLE WITH RESPECT TO NON-LINE
C      1AR CONSTRAINT NUMBER',I3,' VALUE OF CONSTRAINT IS:',1PE20.7)
C      STOP
C      2020 CONTINUE

```

FEAS1460
FEAS1465
FEAS1470
FEAS1480
FEAS1490
FEAS1495
FEAS1500
FEAS1505
FEAS1510
FEAS1520
FEAS1530
FEAS1535
FEAS1540
FEAS1545
FEAS1550
FEAS1560
FEAS1570
FEAS1575
FEAS1580
FEAS1585
FEAS1590
FEAS1595
FEAS1600
FEAS1605
FEAS1610
FEAS1620
FEAS1630
FEAS1640
FEAS1650
FEAS1655
FEAS1660
FEAS1665
FEAS1670
FEAS1675
FEAS1680
FEAS1685
FEAS1690
FEAS1695
FEAS1700
FEAS1705
FEAS1710
FEAS1720
FEAS1730
FEAS1740
FEAS1750
FEAS1760
FEAS1765
FEAS1770


```

C C C STORE NUMBER OF BINDING N. L. CONSTRAINTS.
C C C
C C C JNGI = KOUNT1
C C C GO TO 2032
C C C IF WE GET HERE, THERE ARE NO NON-LINEAR CONSTRAINTS.
C C C
C C C 2030 JNGI = 0
C C C ARE THERE ANY LINEAR CONSTRAINTS
C C C
C C C 2032 IF( ILCNT .LE. 0 ) GO TO 2042
C C C RESET THE COUNTER, THEN CHECK THE LINEAR CONSTRAINTS
C C C
C C C KOUNT1 = 0
C C C DO 2040 N=1, ILCNT
C C C DSUM = 0.0D0
C C C EVALUATE THE N-TH LINEAR CONSTRAINT
C C C
C C C DO 2035 I=1, Ixmax
C C C DEL = DBLE( GLCNST(N,I) )
C C C DBI = DBLE( X(I) )
C C C CHECK FOR NEAR 0 VALUES TO PREVENT UNDERFLOW.
C C C
C C C 2035 IF( DABS(DBI) .LT. 1.0D-39 ) GO TO 2035
C C C IF( DABS(DEL) .LT. 1.0D-39 ) GO TO 2035
C C C DSUM = DSUM + DEL * DBI
C C C CONTINUE
C C C XHI = BL(N)
C C C XX = SNGI(DSUM)
C C C XLO = XHI - EPSLON
C C C IS X INTERIOR WITH RESPECT TO CONSTRAINT N
C C C
C C C IF( XX .LT. XLO ) GO TO 2040
C C C IS X INFEASIBLE WITH RESPECT TO CONSTRAINT N
C C C
C C C IF( XX .GT. XHI ) GO TO 2094
C C C X IS WITHIN EPSILON OF CONSTRAINT N
C C C
C C C KOUNT1 = KOUNT1 + 1
C C C ILBND(KOUNT1) = N

```



```

2094 GO TO 2040
2994 WRITE(NWRITE,2994) N,XX
      FORMAT('O * * * * X IS NOT FEASIBLE WITH RESPECT TO LINEAR
10NCONSTRAINT NUMBER',I3,' VALUE OF CONSTRAINT IS:',IPE20.7)
      STOP
2040 CONTINUE
C
C
C      STORE NUMBER OF BINDING LINEAR CONSTRAINTS
C
C      JNLI = KOUNT1
C      GO TO 2100
C
C      2042 JNLI = 0
C
C      SET UP THE INITIAL XB VECTOR AND BASIS INVERSE
C      VARIABLES:
C      ICROW - # OF ROWS IN THE "CORE" OF THE "A" MATRIX
C      ILPCOL - # OF COLUMNS IN "CORE"
C      IBASIC(I) - LIST OF VARIABLES IN BASIS
C      $BASIC(I) - BASIS STATUS OF ALL VARIABLES
C      XB(I) - VALUE OF VECTORS IN BASIS
C
C      SET VALUES OF ICROW, ICPCROW, ICCOL, ILPCOL
2100 ICROW = JNLI + 2
      ILPCROW = ICROW + IXMAX
      ICCOL = JLO + JHI + 2 * JIN + 2
      ILPCOL = ICCOL + ILPCROW - 1
C
C      XB WILL BE OF FORM (0,0, ..., 0,1, ..., 1)
C
C      DO 2102 I=1,ICROW
C      XB(I) = 0.0EO
2102 CONTINUE
      I1 = ICROW + 1
      IBASIC(I1) = 1
      $BASIC(I1) = .TRUE.
      DO 2104 I=1,ILPCOL
      XB(I) = 1.0EO
2104 CONTINUE
C
C      THE BASIS INVERSE IS INITIALLY THE IDENTITY MATRIX
C
C      DO 2108 I=1,ILPCROW
C      DO 2106 N=1,ILPCROW

```

```

FEAS2080
FEAS2090
FEAS2100
FEAS2110
FEAS2120
FEAS2130
FEAS2135
FEAS2140
FEAS2145
FEAS2150
FEAS2160
FEAS2165
FEAS2170
FEAS2175
FEAS2180
FEAS2190
FEAS2200
FEAS2210
FEAS2220
FEAS2230
FEAS2240
FEAS2250
FEAS2260
FEAS2270
FEAS2280
FEAS2290
FEAS2300
FEAS2310
FEAS2320
FEAS2330
FEAS2335
FEAS2340
FEAS2345
FEAS2350
FEAS2360
FEAS2370
FEAS2380
FEAS2390
FEAS2400
FEAS2410
FEAS2420
FEAS2430
FEAS2435
FEAS2440
FEAS2445
FEAS2450
FEAS2460

```



```

C      BINVRS(N,I) = 0.000
C      SET THE DIAGONAL ELEMENTS TO 1.0
C      IF( N .EQ. I ) BINVRS(I,I) = 1.000
C      CONTINUE
2106 CONTINUE
2108 CONTINUE
C      THE ONLY VARIABLES IN THE BASIS ARE 1 AND THE SLACK VARIABLES
C
C      DO 2110 I=2,ICCOL
C      $BASIS(I) = .FALSE.
2110 CONTINUE
C      I1 = ICCOL + 1
C      DO 2112 I=I1,I1PCOL
C      $BASIS(I) = .TRUE.
C      IBASIC(I-I1+2) = I
2112 CONTINUE
C      BUILD THE "A" MATRIX:
C      WE WILL CALCULATE AND STORE THOSE ELEMENTS OF THE
C      L. P. "A" MATRIX THAT DEPEND ON THE VALUE OF X, AND ON
C      WHICH OF THE CONSTRAINTS. THESE WILL BE STORED IN A MATRIX
C      CALLED "ACORE". THE OTHER ELEMENTS ARE ALL EITHER 0.0 OR
C      1.0 AND DO NOT DEPEND ON X, SO THEY WILL BE GENERATED ONLY
C      WHEN NEEDED.
C      ARE THERE ANY VARIABLES AT THEIR LOWER BOUND
C
C      IF( JLC .LE. 0 ) GO TO 2122
C
C      FOR VARIABLES AT THEIR LOWER BOUND, THE COLUMN IN THE
C      A MATRIX IS OF THE FORM (0,DELFX,DELGI,DELX,DELLIX,DELLX,
C      0, . . . , 1, . . . , 0). IN IAJ(I) KEEP A LIST OF THE VARIABLES
C      IN THE ORDER THEY ARE PLACED IN THE "A" MATRIX
C
C      DO 2125 I=1,JLC
C      N1 = IMIN(I)
C      IAJ(I) = N1
C      ACORE(1,I) = -DELFX(N1,X)
C
C      ARE THERE ANY BINDING NON-LINEAR CONSTRAINTS
C
C      IF( JNGI .LE. 0 ) GO TO 2125
C      DO 2120 N=1,JNGI
C      N2 = IGBIND(N)

```

```

FEAS2470
FEAS2475
FEAS2480
FEAS2485
FEAS2490
FEAS2500
FEAS2510
FEAS2515
FEAS2520
FEAS2525
FEAS2530
FEAS2540
FEAS2550
FEAS2560
FEAS2570
FEAS2580
FEAS2590
FEAS2600
FEAS2610
FEAS2620
FEAS2630
FEAS2640
FEAS2650
FEAS2660
FEAS2670
FEAS2680
FEAS2690
FEAS2700
FEAS2705
FEAS2706
FEAS2710
FEAS2715
FEAS2720
FEAS2730
FEAS2740
FEAS2750
FEAS2760
FEAS2770
FEAS2775
FEAS2780
FEAS2790
FEAS2800
FEAS2805
FEAS2810
FEAS2815
FEAS2820
FEAS2830
FEAS2840

```



```

XX = DELGIX(N1,N2,X)
IF( ABS(XX).LT.1.OE-39 ) XX = 0.OEO
ACORE(N+1,I) = XX
CONTINUE
CONTINUE
C
C
C ARE ANY VARIABLES AT THEIR UPPER BOUND
C
C2122 IF( JHI .LE. 0 ) GO TO 2132
C
FOR VARIABLES AT THEIR UPPER BOUND THE COLUMN OF ACORE IS
OF THE FORM (-DELFX, -DELGX, ...)
DO 2130 I=1,JHI
N1 = IMAX(I)
I1 = JLO + I
IAJ(I1) = -N1
C
C THE MINUS SIGN INDICATES THIS IS AN UPPER BOUND
C COLUMN AND WILL BE USED LATER WHEN GENERATING AJ.
C
C ACORE(1,I1) = DELFX(N1,X)
C
C ARE THERE ANY BINDING NON-LINEAR CONSTRAINTS.
C
C IF( JNGI .LE. 0 ) GO TO 2130
C DO 2126 N=1,JNGI
C N2 = IGBIND(N)
C XX = -DELGIX(N1,N2,X)
C IF( ABS(XX).LT.1.OE-39 ) XX = 0.OEO
C ACORE(N+1,I1) = XX
C
C CONTINUE
C2126 CONTINUE
C
C ARE THERE ANY VARIABLES NOT AT A BOUND.
C
C2132 IF( JIN .LE. 0 ) GO TO 2150
C
FOR VARIABLES NOT AT A BOUND, WE GENERATE TWO COLUMNS, BOUND
ONE LIKE A LOWER BOUND COLUMN, AND ONE LIKE AN UPPER BOUND
COLUMN.
DO 2140 I=1,JIN
N1 = IXIN(I)
I1 = JLO + JHI + 2 * I
IAJ(I1 - 1) = N1
IAJ(I1) = -N1

```


FEAS33250
FEAS33260
FEAS33270
FEAS33275
FEAS33280
FEAS33285
FEAS33290
FEAS33300
FEAS33310
FEAS33320
FEAS33330
FEAS33340
FEAS33350
FEAS33360
FEAS33370
FEAS33380
FEAS33390
FEAS33400
FEAS33410
FEAS33420
FEAS33430
FEAS33440
FEAS33450
FEAS33460
FEAS33465
FEAS33470
FEAS33480
FEAS33490
FEAS33495
FEAS33500
FEAS33505
FEAS33510
FEAS33520
FEAS33530
FEAS33540
FEAS33550
FEAS33560
FEAS33570
FEAS33575
FEAS33580
FEAS33590
FEAS33595
FEAS33600
FEAS33605
FEAS33610
FEAS33615
FEAS33620
FEAS33630

```

C      XX = DELFX(N1,X)
C      ACORE(1,I1-1) = -XX
C      ACORE(1,I1) = XX
C
C      ARE THERE ANY BINDING NON-LINEAR CONSTRAINTS
C
C      IF(JNGI .LE. 0) GO TO 2140
C      DO 2136 N=1,JNGI
C      N2 = IGBIND(N)
C      XX = DELGIX(N1,N2,X)
C      IF(ABS(XX) .LT. 1.0E-39) XX = 0.0
C      ACORE(N+1,I1-1) = XX
C      ACORE(N+1,I1) = -XX
C      2136 CONTINUE
C      2140 CONTINUE
C
C      CALCULATE ZJ - CJ
C      THE ZJ - CJ VALUES FOR THE VARIABLES NOT IN
C      IN THE BASIS. ZJ - ZJ IS EQUAL TO THE VECTOR PRODUCT OF
C      THE FIRST ROW OF THE BASIS INVERSE AND THE COLUMN VECTOR
C      FROM THE "A" MATRIX CORRESPONDING TO THE VARIABLE. THE
C      ZJ - CJ VALUES ARE STORED IN ZJCJ(I)
C
C      ZERO OUT ZJCJ(I)
C
C      2150 DO 2152 I=1,I LPCOL
C      ZJCJ(I) = 0.0E0
C      2152 CONTINUE
C
C      IF 2 IS NOT IN BASIS CALCULATE ZJCJ(2)
C
C      IF( $BASIC(2) ) GO TO 2156
C      DZJ = -1.0D0
C      IEND = JNGI + 2
C      DO 2154 I=2,IEND
C      DZJ = DZJ + BINVR(1,I)
C      2154 CONTINUE
C      ZJCJ(2) = SNGL(DZJ)
C
C      NOW CALCULATE ZJ - CJ FOR THOSE NON-BASIC
C      VARIABLES IN THE "CORE".
C
C      2156 DO 2166 I=3,I COL
C      CALCULATE ZJ - CJ FOR NON-BASIC VARIABLES ONLY
C
C      IF( $BASIC(I) ) GO TO 2166
C      IEND = JNGI + 1

```



```

FEAS3640
FEAS3650
FEAS3660
FEAS3670
FEAS3680
FEAS3690
FEAS3700
FEAS3710
FEAS3720
FEAS3730
FEAS3740
FEAS3750
FEAS3755
FEAS3760
FEAS3770
FEAS3775
FEAS3780
FEAS3790
FEAS3800
FEAS3810
FEAS3820
FEAS3830
FEAS3840
FEAS3850
FEAS3860
FEAS3870
FEAS3875
FEAS3880
FEAS3890
FEAS3895
FEAS3900
FEAS3910
FEAS3920
FEAS3930
FEAS3935
FEAS3940
FEAS3945
FEAS3950
FEAS3960
FEAS3970
FEAS3980
FEAS3990
FEAS4000
FEAS4010
FEAS4020
FEAS4030
FEAS4040
FEAS4050

```

	I2 = I - 2 N1 = IAJ(I2) N2 = IABS(N1) IS = ISIGN(1,N1) DZJ = 0.000 DO 2158 N=1,IEND DBI = BINVRS(1,N+1) IF(DABS(DBI) .LT. 1.0D-39) GO TO 2158 DEL = DBLE(ACORE(N,I2)) IF(DABS(DEL) .LT. 1.0D-39) GO TO 2158 DZJ = DZJ + DEL * DBI CONTINUE IF(JNLI .LE. 0) GO TO 2162 I1 = IEND + 1 DO 2160 N=1,JNLI J1 = ILBIND(N) DBI = BINVRS(1,N+1) IF(DABS(DBI) .LT. 1.0D-39) GO TO 2160 DEL = DBLE(GLCNST(J1,N2)) IF(DABS(DEL) .LT. 1.0D-39) GO TO 2160 DZJ = DZJ + IS * DEL * DBI CONTINUE NOW CALCULATE THE CONTRIBUTION FROM THE S(J) CONSTRAINTS. J1 = ICROW + N2 DZJ = DZJ + BINVRS(1,J1) ZJCJ(I) = SNGL(DZJ) CONTINUE NOW FOR THE NON-BASIC SLACK VARIABLES I1 = ICCOL + 1 DO 2168 I=1,I LPCOL IF(\$BASIC(I)) GO TO 2168 ZJCJ(I) = SNGL(BINVRS(1,I-I1+2)) CONTINUE SELECT COLUMN TO ENTER BASIS. PICK COLUMN WITH MOST NEGATIVE ZJ-CJ. DO NOT CONSIDER COLUMNS ALREADY IN THE BASIS, NOR THOSE WHOSE YJ VECTOR HAVE NO POSITIVE YJ ELEMENTS. IF ALL ZJ-CJ VALUES ARE GREATER THAN OR EQUAL TO ZERO, THEN WE HAVE FOUND THE
--	--


```

C      OPTIMAL SOLUTION FOR THE DIRECTION FINDING PROBLEM.
C 2170 DO 2172 I=1,ILPCOL
      $YJPOS(I) = .TRUE.
2172 CONTINUE
2174 ZCJMIN = ZCJTST
C      FIND THE MOST NEGATIVE ZJCJ(I)
C
C  DO 2176 I=2,ILPCOL
      IF( $BASIC(I) ) GO TO 2176
      IF( ZJCJ(I) .GE. ZCJMIN ) GO TO 2176
      IF( .NOT. $YJPOS(I) ) GO TO 2176
      ZCJMIN = ZJCJ(I)
      IIN = I
C      ENTERING COLUMN IS IIN
C
C      WERE THERE ANY NEGATIVE ZJ-CJ VALUES.
C  IF( ZCJMIN .GE. ZCJTST ) GO TO 2350
C
C      CALCULATE AJ VECTOR
C      COMPUTE THE ENTIRE AJ VECTOR FOR THE ENTERING
C      VARIABLE. CHECK WHICH COLUMN IS ENTERING. IF 2 FORM OF
C      AJ IS (-1,1,0,0,0,0), IF BETWEEN 3 AND ICCOL
C      FROM IS (0,ACORE,0,0,0,0), ALINEAR, 0,0,0,0,0,0)
C      IF GREATER THAN ICCOL, (0,0,0,0,0,0)
C      ZERO OUT THE AJ VECTOR
C
C 2180 DO 2182 I=1,ILPCOL
      AJ(I) = 0.0E0
2182 CONTINUE
      IF( IIN .EQ. 2 ) GO TO 2192
      IF( IIN .LE. ICCOL ) GO TO 2184
      IIN GREATER THAN ICCOL
      I1 = IIN - ICCOL + 1
      GO TO 2220
C      IIN BETWEEN THREE AND ICCOL
C
C 2184 I2 = IIN - 2
      N1 = IAJ(I2)

```

FEAS4060
FEAS4070
FEAS4080
FEAS4090
FEAS4100
FEAS4110
FEAS4115
FEAS4120
FEAS4125
FEAS4130
FEAS4140
FEAS4150
FEAS4160
FEAS4170
FEAS4180
FEAS4185
FEAS4190
FEAS4195
FEAS4200
FEAS4205
FEAS4210
FEAS4215
FEAS4220
FEAS4230
FEAS4240
FEAS4250
FEAS4260
FEAS4270
FEAS4280
FEAS4290
FEAS4300
FEAS4310
FEAS4315
FEAS4320
FEAS4330
FEAS4340
FEAS4350
FEAS4360
FEAS4365
FEAS4370
FEAS4375
FEAS4380
FEAS4390
FEAS4395
FEAS4400
FEAS4405
FEAS4410
FEAS4420


```

IS = ISIGN(I, N1)
N2 = IABS(N1)
IEND = JNCI + 2
DO 2186 I=2, IEND
  AJ(I) = ACORE(I-1, I2)
2186 CONTINUE
C
C      ARE THERE ANY BINDING LINEAR CONSTRAINTS
C
IF( JNLI .LE. 0 ) GO TO 2190
I1 = IEND
DO 2188 I=1, JNLI
  J1 = ILBIND(I)
  AJ(I+1) = IS * GLCNST(J1, N2)
2188 CONTINUE
2190 J1 = ICROW + N2
  AJ(J1) = 1.0EO
  GO TO 2200
C
C      IIN IS TWO
C
2192 AJ(1) = -1.0EO
  IEND = JNCI + 2
  DO 2194 I=2, IEND
    AJ(I) = 1.0EO
  2194 CONTINUE
C
C      CALCULATE JY VECTOR
C      COMPUTE YJ FOR ENTERING VECTOR. YJ IS THE VECTOR
C      PRODUCT OF B INVERSE AND AJ. THEN SELECT THE VARIABLE
C      LEAVING THE BASIS. LEAVING VARIABLE IS THE ONE WITH THE
C      MINIMUM VALUE OF XB(I)/YJ(I) WITH YJ(I) POSITIVE.
C
2200 XLO = CMAX
  IOUT = -1
  DO 2208 I=1, ILPROW
    DSUM = 0.0DO
    DO 2204 N=1, ILPROW
      DBI = BINVR(I, N)
      IF( DABS(DBI) .LT. 1.0D-39 ) GO TO 2204
      DEL = DBLE(AJ(N))
      IF( DABS(DEL) .LT. 1.0D-39 ) GO TO 2204
      DSUM = DSUM + DBI * DEL
    CONTINUE
    YJ(I) = SNGL(DSUM)
    IF( DSUM .LE. 0.0DO ) GO TO 2208
    XX = XB(I)
    IF( XX .EQ. 0.0EO ) GO TO 2206
  2204

```

FEAS4430
FEAS4440
FEAS4450
FEAS4460
FEAS4470
FEAS4480
FEAS4485
FEAS4490
FEAS4495
FEAS4500
FEAS4510
FEAS4520
FEAS4530
FEAS4540
FEAS4550
FEAS4560
FEAS4570
FEAS4580
FEAS4585
FEAS4590
FEAS4595
FEAS4600
FEAS4610
FEAS4620
FEAS4630
FEAS4640
FEAS4650
FEAS4660
FEAS4670
FEAS4680
FEAS4690
FEAS4700
FEAS4710
FEAS4720
FEAS4730
FEAS4740
FEAS4750
FEAS4760
FEAS4770
FEAS4780
FEAS4790
FEAS4800
FEAS4810
FEAS4820
FEAS4830
FEAS4840
FEAS4850
FEAS4860


```

IF( XX.LT. 0.OEO ) GO TO 2208
XX = XX / DSUM
2206 IF( XX.GE. XLO ) GO TO 2208
XLO = XX
IOUT = I
2208 CONTINUE
C
C      NOW CHECK TO SEE IF THERE WERE ANY POSITIVE
C      YJ VALUES. IF NOT GO BACK AND FIND ANOTHER ENTERING
C      VECTOR.
2210 IF( IOUT.GT. 0 ) GO TO 2300
$YJPOS(IIN) = .FALSE.
GO TO 2174
C
C      ENTERING VECTOR IS A UNIT VECTOR. IIN IS
C      GREATER THEN ICCOL.
2220 IOUT = -1
XLO = CMAX
DO 2226 I=1, ILPROW
DSUM = BINVR(I,I)
YJ(I) = SNGL(DSUM)
IF( DSUM.LE. 0.OEO ) GO TO 2226
XX = XB(I)
IF( XX.EQ. 0.OEO ) GO TO 2222
IF( XX.LT. 0.OEO ) GO TO 2226
XX = XX / DSUM
IF( XX.GE. XLO ) GO TO 2226
XLO = XX
IOUT = I
2222 CONTINUE
GO TO 2210
C
C      CALCULATE PHI AND UPDATE XB
C      THE VECTOR PHI IS USED TO GENERATE THE NEW
C      BASIS INVERSE AND XB. PHI(I) IS -YJ(I) / PIVOT ELEMENT
C      YJ(IOUT). PHI(IOUT) IS ( 1 / YJ(IOUT) ) - 1.O.
C      WILL BE STORED IN SAME LOCATION AS YJ. THEN UPDATE
C      XB. NEW XB IS OLD XB + ( 1 / XB(IOUT) ) * PHI
2300 DBI = DBLE(YJ(IOUT))
2301 DBI = 1.OEO / DBI
IF( DABS(DBI).LT. 1.OEO-39 ) DBI = DSIGN(1.OEO-39,DBI)
DO 2310 I=1, ILPROW
IF( I.NE. IOUT ) GO TO 2304
PHI(I) = SNGL(DBI - 1.OEO)
GO TO 2310

```

```

FEAS4870
FEAS4880
FEAS4890
FEAS4900
FEAS4910
FEAS4920
FEAS4925
FEAS4930
FEAS4940
FEAS4950
FEAS4955
FEAS4960
FEAS4970
FEAS4980
FEAS4985
FEAS4990
FEAS5000
FEAS5005
FEAS5010
FEAS5020
FEAS5030
FEAS5040
FEAS5050
FEAS5060
FEAS5070
FEAS5080
FEAS5090
FEAS5100
FEAS5110
FEAS5120
FEAS5130
FEAS5140
FEAS5150
FEAS5155
FEAS5160
FEAS5170
FEAS5180
FEAS5190
FEAS5200
FEAS5210
FEAS5220
FEAS5230
FEAS5240
FEAS5250
FEAS5260
FEAS5270
FEAS5280
FEAS5290

```


FEAS5300
FEAS5310
FEAS5320
FEAS5330
FEAS5340
FEAS5350
FEAS5360
FEAS5370
FEAS5380
FEAS5390
FEAS5400
FEAS5410
FEAS5420
FEAS5430
FEAS5440
FEAS5450
FEAS5460
FEAS5470
FEAS5480
FEAS5490
FEAS5500
FEAS5510
FEAS5515
FEAS5520
FEAS5530
FEAS5540
FEAS5550
FEAS5560
FEAS5570
FEAS5580
FEAS5590
FEAS5600
FEAS5605
FEAS5610
FEAS5615
FEAS5620
FEAS5630
FEAS5640
FEAS5650
FEAS5660
FEAS5665
FEAS5670
FEAS5680
FEAS5685
FEAS5690
FEAS5700
FEAS5710
FEAS5720

```

2304      DEL = DBLE(YJ(I))
          IF( DABS(DEL) .LT. 1.0D-39 ) GO TO 2308
          DEL = -DEL * DBI
          PHI(I) = SNGL(DEL)
          GO TO 2310
2308      PHI(I) = 0.0E0
2310      CONTINUE
          DBI = DBLE( XB(IOUT) )
          IF( DABS(DBI) .LT. 1.0D-39 ) GO TO 2320
          DO 2312 I=1, ILPROW
              DEL = DBLE(PHI(I))
              IF( DABS(DEL) .LT. 1.0D-39 ) GO TO 2312
              DSUM = DBLE(XB(I))
              DSUM = DSUM + DBI * DEL
              XB(I) = SNGL(DSUM)
2312      CONTINUE
          C      CALCULATE NEW BASIS INVERSE
          C      COMPUTE NEW BASIS INVERSE USING PHI. EACH COLUMN
          C      VECTOR IN THE NEW BASIS INVERSE IS EQUAL TO THE SUM OF THE
          C      OLD COLUMN WITH THE PRODUCT OF PHI AND THE COLUMN PIVOT
          C      ELEMENT.
2320      DO 2328 N=1, ILPROW
          DBI = BINVRS(IOUT,N)
          IF( DABS(DBI) .LT. 1.0D-39 ) GO TO 2328
          DO 2324 I=1, ILPROW
              DEL = DBLE(PHI(I))
              IF( DABS(DEL) .LT. 1.0D-39 ) GO TO 2324
              BINVRS(I,N) = BINVRS(I,N) + DBI * DEL
          CONTINUE
2324      CONTINUE
          C      NOW UPDATE THE BASIS LIST.
          C
          I1 = IBASIC(IOUT)
          $BASIC(I1) = .FALSE.
          $BASIC(IIN) = .TRUE.
          IBASIC(IOUT) = IIN
          $INV = .FALSE.
2329      THE INVERSE IS NOT A DIRECT INVERSION.
          C      NOW GO BACK AND CALCULATE THE NEW ZJ-CJ VALUES.
          C
          GO TO 2150
          C      BASIS REINVERSION
          C      THE DIRECTION FINDING PROBLEM HAS REACHED AN

```



```

C      OPTIMAL SOLUTION.  COMPUTE THE BASIS INVERSE DIRECTLY
C      BY INVERTING CURRENT BASIS.  THE FIRST STEP IS TO BUILD
C      THE CURRENT BASIS FROM THE "A" MATRIX AND THEN INVERTING
C      IT DIRECTLY USING THE LIBRARY ROUTINE DMINV.  THE LIBRARY
C      ROUTINE ARRAY HAS BEEN MODIFIED TO DOUBLE PRECISION AND
C      IS TO PACK AND UNPACK BINVRS.  IF SOLUTION IS STILL
C      OPTIMAL AFTER REINVERSION THEN GO ON.
C
2350 IF( $INV ) GO TO 2380
    BINVRS(1,1) = 1.000
    DO 2352 I=2, ILPROW
        BINVRS(I,1) = 0.000
2352 CONTINUE
    DO 2370 I=2, ILPROW
        I1 = IBASIC(I)
        DO 2354 N=1, ILPROW
            BINVRS(N,I) = 0.000
2354 CONTINUE
        IF( I1.NE. 2 ) GO TO 2359
        BINVRS(1,I) = -1.000
        IEND = JNGI + 2
        DO 2358 N=2, IEND
            BINVRS(N,I) = 1.000
2358 CONTINUE
        GO TO 2370
2359 IF( I1.LE. ICCOL ) GO TO 2360
        N1 = I1 - ICCOL + 1
        BINVRS(N1,I) = 1.000
        GO TO 2370
2360 I2 = I1 - 2
        N1 = IAJ(I2)
        IS = ISIGN(1,N1)
        N2 = IABS(N1)
        IEND = JNGI + 2
        DO 2362 N=2, IEND
            BINVRS(N,I) = DBLE(ACORE(N-1,I2))
2362 CONTINUE
        IF( JNLI.EQ. 0 ) GO TO 2366
        N1 = IEND
        DO 2364 N=1, JNLI
            J1 = ILBIND(N)
            BINVRS(N+N1,I) = IS * DBLE(GLCNST(J1,N2))
2364 CONTINUE
        J1 = ICRCW + N2
        BINVRS(J1,I) = 1.000
2366 CONTINUE
        N1 = ILPROW
2370 CALL DARRAY(2,N1,N1,152,152,BINVRS,BINVRS)
2371

```



```

CALL GMINV(BINVRS,N1,DEL,NWORKL,NWORKM)
CALL QARRAY(1,N1,N1,152,BINVRS,BINVRS)
$INV = .TRUE.
GO TO 2150

```

```

      CALCULATE THE DIRECTION S IS THE OPTIMAL SOLUTION TO
      THE L.P. PROBLEM. S(I) EQUALS THE VALUE OF THE
      CORRESPONDING XB(J). IF THE VARIABLE IS NOT IN THE
      BASIS THEN S(I) EQUALS ZERO. IF THE CRIMAL VALUE IS LESS
      THAN DIRECTION SET EPSLON EQUAL TO EPSLON 1 2 AND REDO
      THE DIRECTION, N FINDING PROBLEM. IF THE VALUE OF XDIR IS
      ZERO (OR VERY NEAR ZERO) THERE IS NO USABLE FEASIBLE
      DIRECTION, THUS WE HAVE REACHED THE OPTIMAL SOLUTION.

```

```

2380 IF( XDIR .LE. 1.0E-10 ) GO TO 2650
      IF( XDIR .LT. EPSLON ) GO TO 2390
      DO 2382 I=1,IXMAX

```

```

          S(I) = 0.0E0

```

```

2382 CONTINUE
      DO 2384 I=2,ILPROW
          I1 = IBASIC(I)
          IF( I1 .GT. ICOL ) GO TO 2384
          N1 = IAJ(I1-2)
          N2 = IABS(N1)
          IS = ISIGN(1,N1)
          S(N2) = IS * XB(I)

```

```

2384 CONTINUE
      GO TO 2400
2390 EPSLON = EPSLON / 2.0
      GO TO 2000

```

```

      DETERMINE THE STEP LENGTH
      LONGEST STEP WILL BE DONE IN TWO STAGES. FIRST THE
      FEASIBLE IS DETERMINED SUCH THAT THE NEW POINT IS STILL
      LESS THAN OR EQUAL TO XLAMF). THEN THE STEP LENGTH
      THE FEASIBLE DIRECTION IS DETERMINED. THIS IS DONE BY
      FIND THE POINT THE DERRIVATIVE GOES TO ZERO. IF THE
      STEP LENGTH IS UNBOUNDED THEN THE PROBLEM IS UNBOUNDED.

```

```

      SAVE THE PRESENT POINT

```

```

2400 DO 2402 I=1,IXMAX
          XSAVE(I) = X(I)
2402 CCNTINUE

```

```

FEAS6210
FEAS6220
FEAS6230
FEAS6240
FEAS6250
FEAS6260
FEAS6270
FEAS6280
FEAS6290
FEAS6300
FEAS6310
FEAS6320
FEAS6330
FEAS6340
FEAS6350
FEAS6360
FEAS6370
FEAS6380
FEAS6390
FEAS6400
FEAS6410
FEAS6420
FEAS6430
FEAS6440
FEAS6450
FEAS6460
FEAS6470
FEAS6480
FEAS6490
FEAS6500
FEAS6510
FEAS6520
FEAS6530
FEAS6540
FEAS6550
FEAS6560
FEAS6570
FEAS6580
FEAS6590
FEAS6600
FEAS6610
FEAS6620
FEAS6630
FEAS6640
FEAS6645
FEAS6650
FEAS6660
FEAS6670

```



```

C          SET INITIAL VALUES, $FEAS IS TRUE, LARGEST KNOWN
C          FEASIBLE STEP IS ZERO, FIRST STEP IS ONE.
C
C          $FEAS = .TRUE.
C          XLAMDA = 1.0EO
C          XLAMF = 0.0EO
C
C          MOVE TO THE NEW POINT.
C
2404 DO 2408 I=1,IXMAX
C      XX = S(I)
C      IF( ABS(XX) .LT. 1.0E-3S ) GO TO 2406
C      X(I) = XSAVE(I) + XX * XLAMDA
C
C      IS THE NEW POINT UNBOUNDED.
C
C      IF( X(I) .GT. CMAX / 2.0 ) GO TO 2452
C      GO TO 2408
C      X(I) = XSAVE(I)
2406 CONTINUE
C
C      NOW CHECK FEASIBILITY OF THE NEW POINT.
C      FIRST CHECK BOUNDS ON X(I).
C
C      DC 2412 I=1,IXMAX
C      XX = X(I)
C      XLO = XMIN(I)
C      XHI = XMAX(I)
C      IF( XX .LT. XLO ) GO TO 2430
C      IF( XX .GT. XHI ) GO TO 2430
2412 CONTINUE
C
C      IF THERE ANY NON-LINEAR CONSTRAINTS, CHECK
C      THEM FOR FEASIBILITY
C
C      IF( IGCNT .LE. 0 ) GO TO 2418
C      DO 2416 N=1,IGCNT
C      XX = GIX(N,X)
C      XHI = BC(N)
C      IF( XX .GT. XHI ) GO TO 2430
2416 CONTINUE
C
C      IF THERE ARE LINEAR CONSTRAINTS, CHECK THEM.
C
2418 IF( ILCNT .LE. 0 ) GO TO 2429
2420 DO 2428 N=1,ILCNT
C      DSUM = 0.0DO

```



```

DO 2422 I=1,IXMAX
  DBI = DBLE(X(I))
  IF( DABS(DBI) .LT. 1.0D-39 ) GO TO 2422
  DEL = DBLE(GLCNST(N,I))
  IF( DABS(DEL) .LT. 1.0D-39 ) GO TO 2422
  DSUM = DSUM + DEL * DBI
  CONTINUE
2422 XX = SNGL(DSUM)
  XHI = BL(N)
  IF( XX .GT. XHI ) GO TO 2430
2428 CONTINUE
C
C      THE NEW POINT IS FEASIBLE.  IF IT HAS A LARGER
C      VALUE FOR THE OBJECTIVE FUNCTION, TAKE ANOTHER
C      STEP, OTHERWISE GO ON TO STAGE TWO.
C
2429 IF( FX(X) .GT. FOFX ) GO TO 2440
  XLAMF = XLAMDA
  GO TO 2450
C
C      IF WE GET HERE POINT IS NOT FEASIBLE.
C
2430 XLAMNF = XLAMDA
  XLAMDA = ( XLAMNF + XLAMF ) / 2.0
  $FEAS = .FALSE.
  IF( XLAMDA .LT. STPMIN ) GO TO 2700
  IF( ( XLAMNF - XLAMF ) .LT. STPMIN ) GO TO 2450
C
C      NOW TRY A SMALLER XLAMDA.
C
  GO TO 2404
C
C      POINT IS FEASIBLE, WAS PREVIOUS POINT FEASIBLE
2440 IF( $FEAS ) GO TO 2444
  XINC = XLAMNF - XLAMF
  IF( XINC .LT. STPMIN ) GO TO 2450
  XLAMF = XLAMDA
  XLAMDA = ( XLAMNF + XLAMF ) / 2.0
  GO TO 2404
2444 IF( XLAMDA .GE. CMAX / 2.0 ) GO TO 2448
  XLAMF = XLAMDA
  XLAMDA = XLAMDA * 2.0
  GO TO 2404
2448 XLAMF = XLAMDA
  GO TO 2452
C
C      NOW FIND THE STEP LENGTH LESS THAN OR EQUAL

```



```

C      TO XLAMF THAT MAXIMIZE F(X) ALONG THE DIRECTION S.
C      2450 $BOUND = .TRUE.
C      XLAMDA = XLAMF
C      GO TO 2454
C
C      IF WE GET HERE, FIRST STAGE STEP LENGTH WAS
C      UNBOUNDED.
C
C      2452 $BOUND = .FALSE.
C
C      XONE WILL CONTAIN THE LOWER BOUND OF THE STEP
C      LENGTH, SLOPE OF F(X) IS POSITIVE AT XONE. XTWO
C      WILL BE UPPER BOUND, SLOPE NEGATIVE, WE WANT TO
C      FIND THE POINT WITH ZERO SLOPE, WHICH WILL BE
C      THE MAX OF F(X) ALONG THE DIRECTION S.
C
C      2454 DO 2460 I=1,IXMAX
C      XX = S(I)
C      XONE(I) = XSAVE(I)
C      IF( ABS(XX) ) .LT. 1.0E-39 ) GO TO 2458
C      X(I) = XSAVE(I) + XX * XLAMDA
C      GO TO 2460
C      2458 X(I) = XSAVE(I)
C      CONTINUE
C      DSUM = 0.0D0
C      XLAMNF = 0.0E0
C      DO 2464 N=1,IXMAX
C      DEL = DBLE(S(N))
C      IF( DABS(DEL) ) .LT. 1.0D-39 ) GO TO 2464
C      DBI = DBLE(DELFX(N,X))
C      IF( DABS(DBI) ) .LT. 1.0D-39 ) GO TO 2464
C      DSUM = DSUM + DEL * DBI
C      2464 CONTINUE
C      IF( DSUM .GT. 0.0D0 ) GO TO 2476
C      BB = SNGL(DSUM)
C      DSUM = 0.0D0
C      XLAMUP = XLAMDA
C      DO 2468 N=1,IXMAX
C      DBI = DBLE(S(N))
C      IF( DABS(DBI) ) .LT. 1.0D-39 ) GO TO 2468
C      DEL = DBLE(DELFX(N,XONE))
C      IF( DABS(DEL) ) .LT. 1.0D-39 ) GO TO 2468
C      DSUM = DSUM + DEL * DBI
C      2468 CONTINUE
C      AA = SNGL(DSUM)
C
C      THE NEXT STEP LENGTH TO TRY IS THAT STEP WHICH

```



```

C          WOULD HAVE A ZERO SLOPE IF THE SLOPE WAS A LINEAR
C          FUNCTION OF STEP LENGTH.
C          2470 XLAMDA = ( BB * XLAMNF - AA * XLAMUP ) / ( BB - AA )
C
C          WE HAVE A NEW XTWO, UPPER BOUND.
C
C          DO 2474 I=1,IXMAX
C             XX = S(I)
C             XTWO(I) = X(I)
C             IF( ABS(XX) ) .LT. 1.0E-39 ) GO TO 2472
C             X(I) = XSAVE(I) + XX * XLAMDA
C             GO TO 2474
C             X(I) = XSAVE(I)
C          CONTINUE
C          2472 2474 GO TO 2480
C
C          IF WE GET HERE F(X) IS INCREASING, BUT WE ARE
C          AT A CONSTRAINT, IF PROB IS STILL BOUNDED GO ON
C          TO NEXT ITERATION, OTHERWISE WE ARE DONE.
C
C          2476 IF( FX(X) .GT. FX(XSAVE) ) GO TO 2478
C          XLAMDA = XLAMDA / 2.0
C          GO TO 2454
C          2478 IF( $BOUND ) GO TO 2500
C          GO TO 2800
C          2480 DSUM = 0.0D0
C          DO 2482 N=1,IXMAX
C             DBI = DBLE(S(N))
C             IF( DABS(DBI) ) .LT. 1.0D-39 ) GO TO 2482
C             DEL = DBLE(DELFX(N,X))
C             IF( DABS(DEL) ) .LT. 1.0D-39 ) GO TO 2482
C             DSUM = DSUM + DEL * DBI
C          CONTINUE
C          2482 IF( DSUM .EQ. 0.0D0 ) GO TO 2500
C          IF( DSUM .LT. 0.0D0 ) GO TO 2487
C
C          IF NEXT STEP IS TOO SMALL WE ARE DONE.
C
C          IF( ( XLAMUP - XLAMDA ) .LT. STPMIN ) GO TO 2490
C          AA = SNGL(DSUM)
C          XLAMNF = XLAMDA
C          XLAMDA = ( BB * XLAMNF - AA * XLAMUP ) / ( BB - AA )
C
C          WE HAVE A NEW XONE, LOWER BOUND.
C
C          DO 2486 I=1,IXMAX
C             XX = S(I)

```



```

XONE(I) = X(I)
IF( ABS(XX) .LT. 1.0E-39 ) GO TO 2485
X(I) = XSAVE(I) + XX * XLAMDA
GO TO 2486
2485 X(I) = XSAVE(I)
2486 CONTINUE
GO TO 2480

C
C
C      IF NEXT STEP IS TOO SMALL WE ARE DONE.
2487 IF( ( XLAMDA - XLAMNF ) .LT. STPMIN ) GO TO 2488
      BB = SNGL(DSUM)
      XLAMUP = XLAMDA
      GO TO 2470
2488 IF( FX(X) ) .GT. FX(XONE) ) GO TO 2500
      DO 2489 I=1, IXMAX
        X(I) = XONE(I)
2489 CONTINUE
      GO TO 2500
2490 IF( FX(X) ) .GT. FX(XTWO) ) GO TO 2500
      DO 2492 I=1, IXMAX
        X(I) = XTWO(I)
2492 CONTINUE

C
C
C      WE HAVE A NEW POINT, STORED IN X.  GET SET
      FOR NEXT ITERATION
2500 XX = FX(X)
      AA = ABS(FOFX)
      IF( AA .EQ. 0.0E0 ) AA = 1.0E-02
      AX = ( ( XX - FOFX ) / AA ) * 1.0E02
      FOFX = XX
      IF( AX .LT. PERCENT ) GO TO 2600
      IF( ITER .GT. ITMAX ) GO TO 2550
      WRITE(NWRITE,2900) ITER, FOFX, AX
2900 FORMAT( 'O ITERATION NUMBER:', I6, '/', I6X, 'F(X) = ', IPE15.6, ' ; IMPROVE
1MENT THIS ITERATION:', OPF10.4, ' PERCENT.' )
      ITER = ITER + 1
      WRITE(NWRITE,2902) ( I, X(I), I=1, IXMAX )
      GO TO 2000
2550 WRITE(NWRITE,2901) ITER, FOFX
2901 FORMAT( 'O PROGRAM TERMINATED AFTER', I6, ' ITERATIONS.', '/', 'O THE PRES
1ENT VALUE OF THE OBJECTIVE FUNCTION IS:', IPE15.6 )
2555 WRITE(NWRITE,2902) ( I, X(I), I=1, IXMAX )
2902 FORMAT( 'O', 5X, 'THE POINT X( ) IS:', '/', '( 'O', 5X, 4('X(', I2, '), ', G15.7
1, ' ; ) / ) )
      WRITE(NWRITE,905)
905 FORMAT( '1, ) )

```

FEAS7930
FEAS7940
FEAS7950
FEAS7960
FEAS7970
FEAS7980
FEAS7990
FEAS7996
FEAS7997
FEAS7998
FEAS8000
FEAS8010
FEAS8020
FEAS8030
FEAS8040
FEAS8050
FEAS8060
FEAS8070
FEAS8080
FEAS8090
FEAS8100
FEAS8110
FEAS8120
FEAS8125
FEAS8130
FEAS8140
FEAS8145
FEAS8150
FEAS8158
FEAS8160
FEAS8170
FEAS8180
FEAS8190
FEAS8200
FEAS8210
FEAS8220
FEAS8230
FEAS8235
FEAS8240
FEAS8250
FEAS8260
FEAS8270
FEAS8280
FEAS8290
FEAS8295
FEAS8298
FEAS8299


```

2600 STOP
2601 WRITE(NWRITE,2903) FOFX, AX, PERCENT
2602 FORMAT('O PROGRAM TERMINATED BECAUSE OF INSUFFICIENT IMPROVEMENT
2603 IN OBJECTIVE FUNCTION.',/,',O VALUE OF OBJECTIVE FUNCTION:',IPE20.7,
2604 2/O IMPROVEMENT LAST ITERATION WAS',OPF10.4,', PERCENT, WHICH IS LE
2605 3SS THAN THE LIMIT ',F10.4)
2610 WRITE(NWRITE,2902) ( I, X(I), I=1, Ixmax )
2611 WRITE(NWRITE,905)
2620 STOP
2650 WRITE(NWRITE,2907) FOFX
2651 FORMAT('O OPTIMAL SOLUTION:',/,',O THE OPTIMAL VALUE OF THE OBJECTI
2652 IVE FUNCTION IS:',IPE15.6)
2653 WRITE(NWRITE,2902) ( I, X(I), I=1, Ixmax )
2654 WRITE(NWRITE,905)
2660 STOP
2700 WRITE(NWRITE,2904) FOFX
2701 FORMAT('O PROGRAM TERMINATED. LENGTH OF LAST STEP WAS LESS THAN
2702 THE MINIMUM STEP.',/,',O PRESENT VALUE OF THE OBJECTIVE FUNCTION:',
2703 2IPE15.6)
2710 GO TO 2610
2800 WRITE(NWRITE,2905)
2801 FORMAT('O PROBLEM IS UNBOUNDED. LAST POINT AND DIRECTION ARE:')
2802 WRITE(NWRITE,2902) ( I, XSAVE(I), I=1, Ixmax )
2803 WRITE(NWRITE,2905) ( I, S(I), I=1, Ixmax )
2804 FORMAT('O',5X,'LAST DIRECTION:',/,',O',5X,4('S(',I2,')'; ',G20.6,'
2805 1,')/))
2806 WRITE(NWRITE,905)
2807 STOP
2808 END
FEAS83300
FEAS83310
FEAS83320
FEAS83330
FEAS83340
FEAS83350
FEAS83360
FEAS83365
FEAS83370
FEAS83372
FEAS83373
FEAS83374
FEAS83375
FEAS83376
FEAS83377
FEAS83380
FEAS83390
FEAS83391
FEAS83392
FEAS8400
FEAS8410
FEAS8420
FEAS8430
FEAS8440
FEAS8450
FEAS8460
FEAS8465
FEAS8470
FEAS9000

```



```

C      ARE THERE ANY NON-LINEAR CONSTRAINTS
C
C      IF( IGCNT .LE. 0 ) GO TO 2030
C      DO 2020 N=1, IGCNT
C      XX = GIX(N,X)
C      XHI = BC(N)
C
C      IS X INFEASIBLE WITH RESPECT TO CONSTRAINT N
C
C      IF( XX .GT. XHI ) GO TO 2092
C      WRITE(NWRITE,2091) N, XX
C      FORMAT('0 VALUE OF NON-LINEAR CONSTRAINT NUMBER',I3,' ', IS:',
2091 1 IPE15.7)
C      GO TO 2020
C      WRITE(NWRITE,2992) N, XX
C      FORMAT('0 * * * * * X IS NOT FEASIBLE WITH RESPECT TO NON-LINE
2992 1 AR CONSTRAINT NUMBER',I3,' VALUE OF CONSTRAINT IS:',IPE20.7)
C      CONTINUE
C      ARE THERE ANY LINEAR CONSTRAINTS
C
C      DO 2030 IF( ILCNT .LE. 0 ) GO TO 2042
C      CHECK THE LINEAR CONSTRAINTS.
C
C      DO 2040 N=1, ILCNT
C      DSUM = 0.000
C      EVALUATE THE N-TH LINEAR CONSTRAINT
C
C      DO 2035 I=1, IXMAX
C      DEL = DBLE( GLCNST(N,I) )
C      DBI = DBLE( X(I) )
C      CHECK FOR NEAR 0 VALUES TO PREVENT UNDERFLOW.
C
C      IF( DABS(DBI) .LT. 1.0D-39 ) GO TO 2035
C      IF( DABS(DEL) .LT. 1.0D-39 ) GO TO 2035
C      DSUM = DSUM + DEL * DBI
C      CONTINUE
C      XHI = BL(N)
C      XX = SNGL(DSUM)
C
C      IS X INFEASIBLE WITH RESPECT TO CONSTRAINT N
C
C      IF( XX .GT. XHI ) GO TO 2094
C      WRITE(NWRITE,2993) N, XX
C      FORMAT('0 THE VALUE OF LINEAR CONSTRAINT NUMBER',I3,' ', IS:',IPE15
2035 1

```

CHE01450
 CHE01460
 CHE01470
 CHE01480
 CHE01490
 CHE01500
 CHE01510
 CHE01520
 CHE01530
 CHE01540
 CHE01550
 CHE01560
 CHE01561
 CHE01570
 CHE01580
 CHE01590
 CHE01600
 CHE01610
 CHE01620
 CHE01630
 CHE01640
 CHE01650
 CHE01660
 CHE01670
 CHE01680
 CHE01690
 CHE01700
 CHE01710
 CHE01720
 CHE01730
 CHE01740
 CHE01750
 CHE01760
 CHE01770
 CHE01780
 CHE01790
 CHE01800
 CHE01810
 CHE01820
 CHE01830
 CHE01840
 CHE01850
 CHE01860
 CHE01870
 CHE01880
 CHE01890
 CHE01900
 CHE01910


```

1.6) GO TO 2040
2094 WRITE(NWRITE,2994) N,XX
2994 FORMAT('O * * * * X IS NOT FEASIBLE WITH RESPECT TO LINEAR C
2040 CONSTRAINT NUMBER',I3,' VALUE OF CONSTRAINT IS:',IPE20.7)
2040 CCNTINUE
C
2042 WRITE(NWRITE,2942)
2942 FORMAT('O THE BOUNDS, AND ALL THE NON-LINEAR AND LINEAR CONSTRAINTS
IS HAVE BEEN CHECKED.//',NEXT THE VALUES OF THE PARTIAL DERIVAT
2IVES OF THE OBJECTIVE FUNCTION AND THE CONSTRAINTS ARE LISTED.')
```

```

C
DO 2050 N=1,IXMAX X)
XX = DELFX(N,2950) N, XX
WRITE(NWRITE,2950) N, XX
2950 FORMAT('O THE VALUE OF DELFX(',I2,', X) IS:',IPE15.6)
2050 CCNTINUE
IF( IGCNT.EQ.0 ) GO TO 3000
DO 2060 N1=1,IXMAX
XX = DELGIX(N,N1,X)
WRITE(NWRITE,2960) N, N1, XX
2060 CCNTINUE
2960 FORMAT('O THE VALUE OF DELGIX(',I2,',',I2,', X) IS:',IPE15.6)
3000 STOP
END
```

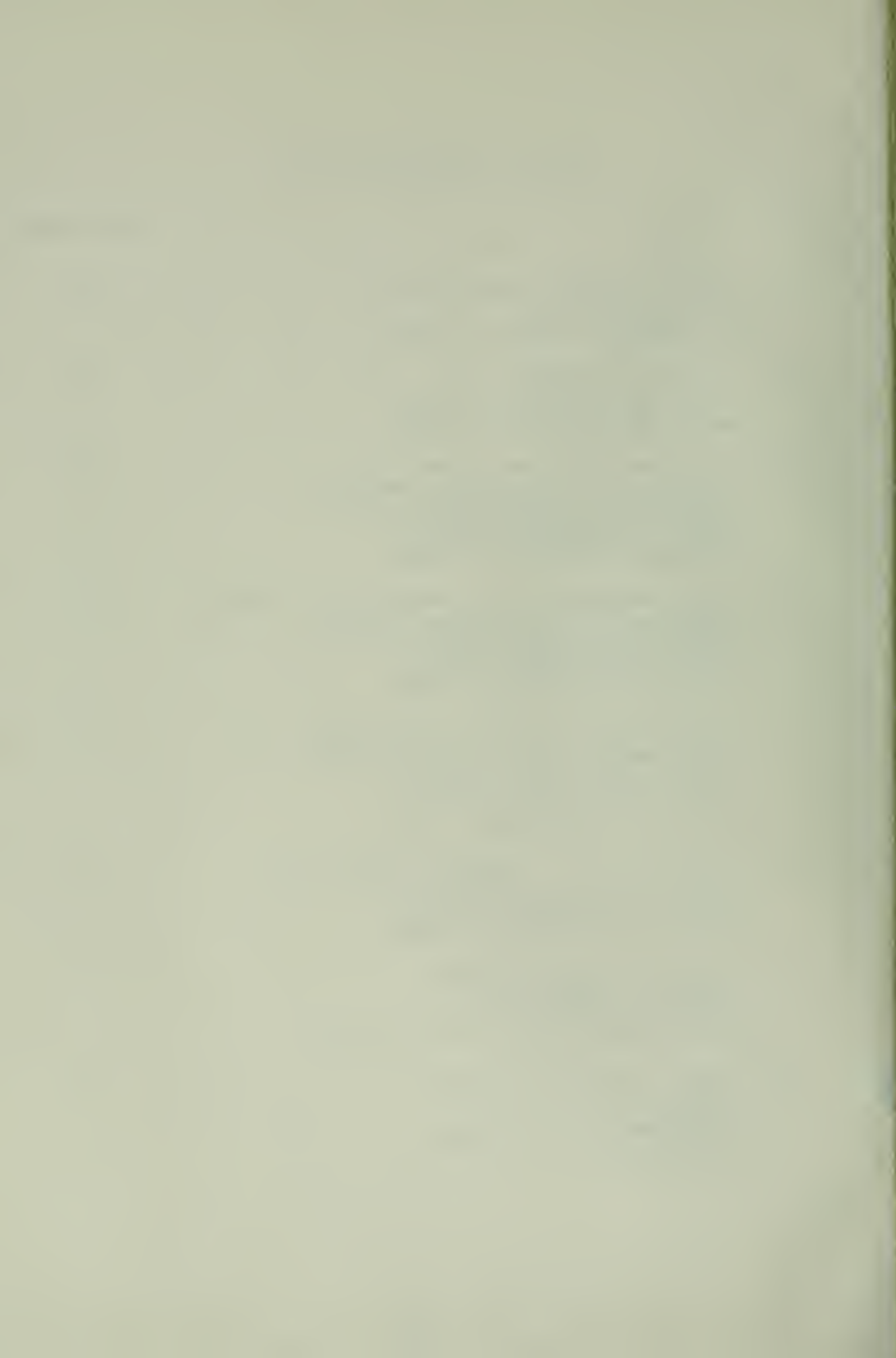
CHE01920
CHE01930
CHE01940
CHE01950
CHE01960
CHE01970
CHE01980
CHE01990
CHE02000
CHE02010
CHE02020
CHE02030
CHE02040
CHE02050
CHE02060
CHE02070
CHE02080
CHE02090
CHE02100
CHE02110
CHE02120
CHE02130
CHE02140
CHE02150
CHE02160
CHE02170

LIST OF REFERENCES

1. Zoutendijk, G., Methods of Feasible Directions, Elsevier, 1960.
2. Arden, B. W. and Astill, K. N., Numerical Algorithms: Origins and Applications, Addison-Wesley, 1970.
3. Hadley, G., Linear Programming, Addison-Wesley, 1962.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3. Department Chairman, Code 55 Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	2
4. Assoc Professor G. T. Howard, Code 55Hk Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	10
5. Assoc Professor W. M. Raike, Code 55Rj Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
6. Professor D. G. Williams, Code 0211 Director, W. R. Church Computer Center Naval Postgraduate School Monterey, California 93940	3
7. LCDR J. Douglas Harrison Commander THIRD Fleet Fleet Post Office San Francisco, California 96610	1
8. Chief of Naval Personnel Pers 11b Department of the Navy Washington, D. C. 20370	1



Thesis
H29329
c.1

Harrison

155153

A computer code for
solving medium sized
non-linear programming
problems by the method
of feasible directions.

Thesis
H29329
c.1

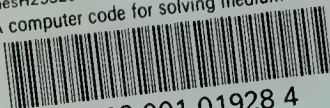
Harrison

155153

A computer code for
solving medium sized
non-linear programming
problems by the method
of feasible directions.

thesH29329

A computer code for solving medium sized



3 2768 001 01928 4

DUDLEY KNOX LIBRARY